



US006941377B1

(12) United States Patent
Diamant et al.**(10) Patent No.: US 6,941,377 B1**
(45) Date of Patent: Sep. 6, 2005**(54) METHOD AND APPARATUS FOR SECONDARY USE OF DEVICES WITH ENCRYPTION****(75) Inventors:** Nimrod Diamant, Kfar-Saba (IL); Marcus Caleschietta, Beaverton, OR (US)**(73) Assignee:** Intel Corporation, Santa Clara, CA (US)**(*) Notice:** Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.**(21) Appl. No.: 09/476,613****(22) Filed: Dec. 31, 1999****(51) Int. Cl.⁷ G06F 15/16****(52) U.S. Cl. 709/230; 709/250****(58) Field of Search 709/230, 105, 709/227, 250; 370/466, 463; 713/200-201****(56) References Cited****U.S. PATENT DOCUMENTS**

5,490,252 A *	2/1996	Macara et al.	709/249
5,963,720 A *	10/1999	Grossman	709/250
6,055,236 A *	4/2000	Nessett et al.	370/389
6,108,562 A *	8/2000	Rydbeck et al.	455/552.1
6,182,149 B1 *	1/2001	Nessett et al.	709/247
6,219,697 B1 *	4/2001	Lawande et al.	709/221
6,222,855 B1 *	4/2001	Kimber et al.	370/463
6,243,395 B1 *	6/2001	Fujimori et al.	370/395.1
6,253,321 B1 *	6/2001	Nikander et al.	713/160
6,314,525 B1 *	11/2001	Mahalingham et al.	714/4
6,321,323 B1 *	11/2001	Nugroho et al.	712/34
6,324,583 B1 *	11/2001	Sivens	709/227
6,424,621 B1 *	7/2002	Ramaswamy et al.	370/230
6,438,678 B1 *	8/2002	Cashman et al.	712/34
6,446,192 B1 *	9/2002	Narasimhan et al.	712/29
6,560,630 B1 *	5/2003	Vepa et al.	718/105
6,590,897 B1 *	7/2003	Lauffenburger et al.	370/395.6
6,708,273 B1 *	3/2004	Ober et al.	713/189
6,799,223 B1 *	9/2004	Yamamoto	709/250

2003/0074473 A1 * 4/2003 Pham et al. 709/246

OTHER PUBLICATIONS

Kent et al., RFC 2401 entitled "Security Architecture for the Internet Protocol", Nov. 1998.*

Keromytis, A.D., "Implementing IPsec" Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE, vol. 3, Nov. 3-8, 1997, pp.: 1948-1952.*

Chappell, B.L., "IP security impact on system performance in a distributed real-time environment", Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE, Dec. 1-3, 1999, pp.: 218-219.*

Ogawa et al., "Smart Cluster Network (SCnet): Design of High Performance Communications for SAN", Cluster Computing, 1999, 1st IEEE Computer Society International Workshop on Dec. 2-3 1999, pp. 71-80.*

* cited by examiner

Primary Examiner—Jack B. Harvey*Assistant Examiner*—Douglas Blair*(74) Attorney, Agent, or Firm*—Steven D. Yates**(57) ABSTRACT**

The invention provides for utilizing abilities of network interfaces, such as embedded encryption support, or access to such encryption support, so as to extend support for such abilities to network interfaces or other devices lacking such ability. In one configuration, a non-homogeneous team of network interfaces is presented to a protocol stack as being a homogeneous team, by having network interfaces lacking a particular ability be backed up by team member supporting the ability. Various methods may be applied to distribute the work load of backing up network interface according to an operation mode of the team. For example, when operating in load balancing mode, performing backup services is balanced across the team, whereas in a fault tolerant mode, processing may be first given to non-primary network interfaces.

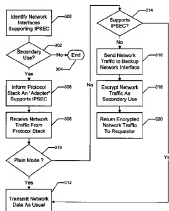
26 Claims, 7 Drawing Sheets

FIG. 1

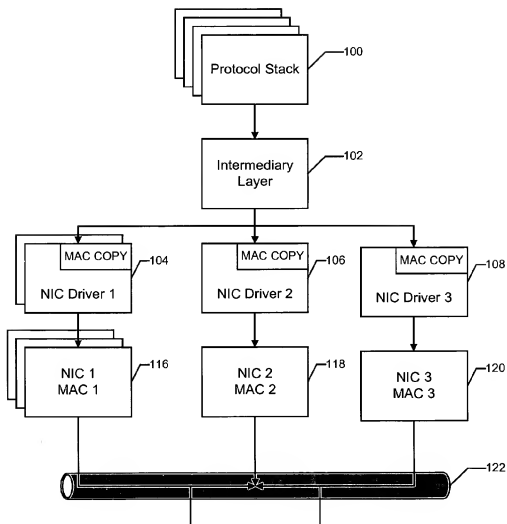


FIG. 2

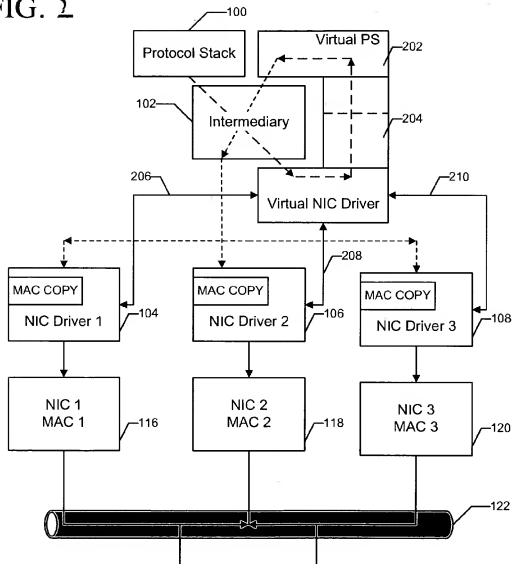


FIG. 3

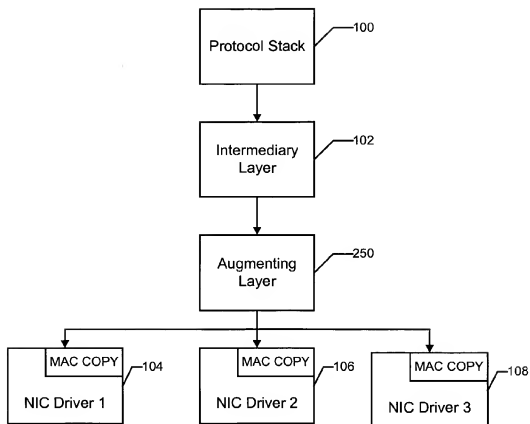


FIG. 4

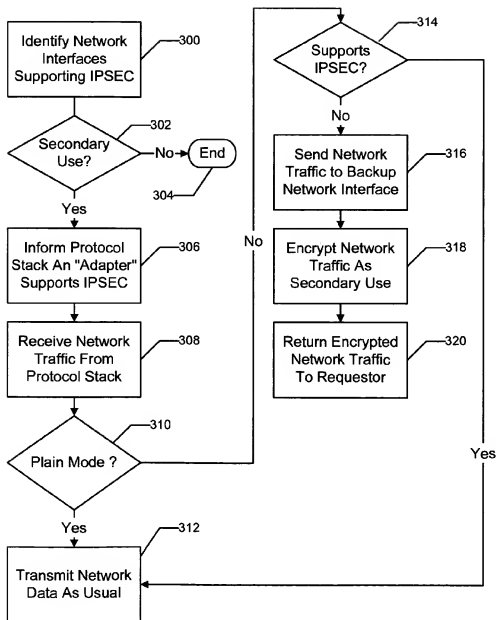


FIG. 5

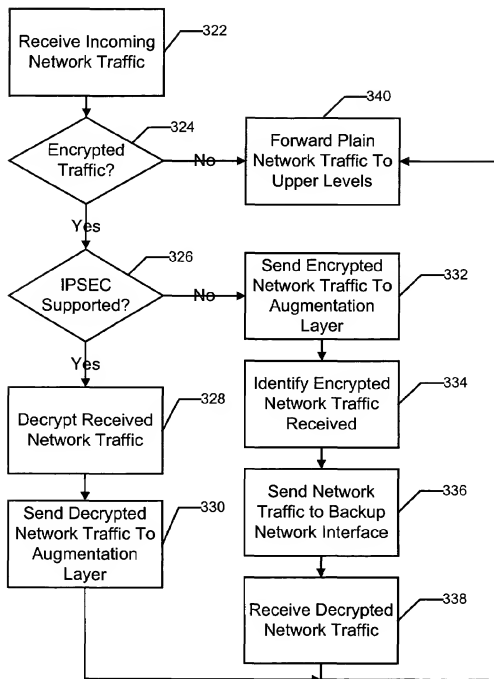


FIG. 6

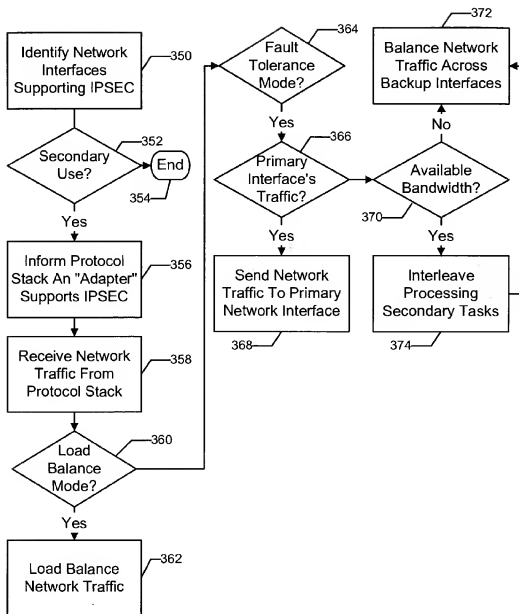
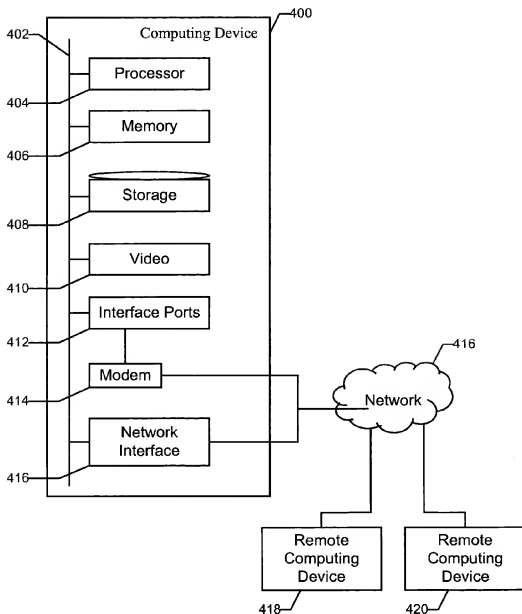


FIG. 7



1 METHOD AND APPARATUS FOR SECONDARY USE OF DEVICES WITH ENCRYPTION

FIELD OF THE INVENTION

The invention generally relates to secondary use of encryption devices, and more particularly to utilizing encryption hardware in network interface cards to provide encryption support for network interfaces lacking encryption support, and to provide parallel execution of encryption tasks by spreading such tasks across multiple network interface card encryption processors.

BACKGROUND

In conventional environments, encryption and decryption is usually performed by software. Due to the complexity involved with performing encryption, the host processor can be greatly burdened with this encryption task. This task burden is commensurate with the degree of security provided by the encryption. Unfortunately, availability of very fast computing hardware has allowed criminals to realistically apply brute-force decryption techniques to private data. Previously, typical encryption methods, such as the Data Encryption Standard (DES), used encryption key lengths of around 40-60 bits, and were considered secure.

But, as several well-publicized contests by RSA Data Security Inc. have shown, such key lengths can be compromised in a matter of days or hours. Thus, to compensate, longer key lengths (e.g., 1024 bits or higher) and more complex encryption schemes are required. This then increases the burden on the host processing system.

Such security concerns have driven efforts to provide secure networking protocols, such as Internet Protocol (IP) security, or IPSEC, promulgated by the Internet Engineering Task Force (IETF) (see IPSEC proposals at Internet location <http://www.ietf.org/ids/wg/ipsec.html>.) This modified IP protocol refers to encrypting IP data traffic with large key lengths and complex encryption algorithms. But, as noted above, such keys and algorithms burdens a host processor already responsible for general networking overhead, and overhead from executing other host processes.

SUMMARY

The invention provides utilization of multiple network interfaces. Network data is received for transmission by a first network interface according to a protocol. It is determined whether the first network interface supports the protocol. If the protocol is not supported, then the network data is provided to a second network interface for processing according to the protocol. The processed network data is transmitted by the first network interface.

BRIEF DESCRIPTION OF THE DRAWINGS

Features and advantages of the invention will become apparent to one skilled in the art to which the invention pertains from review of the following detailed description and claimed embodiments of the invention, in conjunction with the drawings in which:

FIG. 1 illustrates a typical network communication configuration.

FIG. 2 illustrates a low-level view of one embodiment for providing additional networking features not ordinarily supported by a network interface.

FIG. 3 illustrates the logical structure of a FIG. 2 embodiment.

FIG. 4 is a flowchart for using a non-homogeneous team of network adapters as a homogeneous team supporting a desired protocol or functionality.

FIG. 5 is a flowchart illustrating one embodiment for processing receipt of network traffic sent according to FIG. 4.

FIG. 6 illustrates one embodiment of using a team of network interfaces to boost secondary use encryption by distributing an encryption task across multiple team members.

FIG. 7 illustrates a suitable computing environment in which certain aspects the claimed invention may be practiced.

DETAILED DESCRIPTION

In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be understood by those skilled in the art that the present invention may be practiced without these specific details. In other instances well known methods, procedures, components, and circuits have not been described in detail so as not to obscure the present invention.

The increasing burden of performing secured encryption with long keys and complex algorithms provides an opportunity for developers to provide a way to offload encryption burdens from a host's processor. In one embodiment, network interface developers couple an encryption processor with their network interfaces that can be used to encrypt/decrypt network traffic, as well as to provide encryption services to external hardware and processes. In one embodiment, a driver for network interfaces provides access to the encryption hardware, so as to allow external hardware and processes to avoid relying on software encryption methods. Note that the encryption processor may be physically packaged with a network interface, e.g., by way of an encryption application specific integrated circuit (ASIC) (or equivalent) on a network interface, or packaged separately and communicatively thereto.

FIG. 1 illustrates a typical network communication configuration, in which a protocol stack **100** is in communication with an intermediary layer **102** (e.g., LSL or NDIS). There may, as illustrated, be several protocol stacks **100**. It is assumed there is only a single protocol stack and a single intermediary layer. The protocol stack corresponds to typical networking protocols such as TCP, IP, SPX, IPX, NetBios, Netbeui, AppleTalk, X.400, and the like. The intermediary layer **102** is bound to the protocol stack, and helps route network traffic.

The intermediary layer is in communication with multiple network interface card base drivers **104-108**. As shown, instances of a single base driver **104** can be managing multiple network interfaces (three such interfaces are illustrated as a stack of interfaces **116**). For presentation clarity, it is assumed each base driver communicates with a single network interface. Note that although network interface cards, or "NICs", are shown, the term NIC is meant to include other input/output interfaces for alternate network configurations, such networks effected over serial/parallel port connections, Universal Serial Bus (USB) links, IEEE 1394 FireWire link, and the like.

In the illustrated configuration, the intermediary **102** appears to the stack **100** as a multiplexer to the different base drivers. The stack and base drivers are bound to the inter-

3

4

mediary, resulting in network data received by the protocol stack being routed to the intermediary. The intermediary becomes responsible for forwarding the network data on to an appropriate base driver **104-108** which is then responsible for transfer of the data to the NIC hardware **116-120** for delivery over a network connection **122**.

On data reception over the network **122**, all NICs see the data, but only the NIC hardware with the appropriate matching MAC filter responds to the incoming data. If a NIC accepts network data, it is forwarded to its driver, which in turn forwards it to the intermediary layer which multiplexes the data to an appropriate protocol stack.

The intermediary layer is capable of accepting many upper-layer protocol stacks, in addition to multiple drivers below it. Although not provided by present networking environments, this ability of the intermediary layer provides an opportunity for allowing transparent fail-over, load-balancing, and support for new network protocols and features, without changing existing base drivers **104-108** for current network interfaces **116-120**.

FIG. 2 illustrates a low-level view of one embodiment for providing additional networking features not ordinarily supported by a network interface. FIG. 3 illustrates the logical structure of the FIG. 2 embodiment. In effect, FIG. 2 provides an "augmenting layer" **250** between a traditional intermediary layer **102** and its network interface drivers **104, 106, 108**, providing opportunity to augment network interface drivers with functionality not originally planned for network interfaces **116, 118, 120**.

In one embodiment, an augmentation layer **250** is implemented by "surrounding" an intermediary layer **102** with a virtual protocol stack **202** and a virtual NIC driver **204**. However, it will be appreciated by those skilled in the art that other configurations may be used to achieve a similar augmentation layer effect. (Note that this figure is highly abstracted to show general structure, and not implementation details.) A protocol stack **100**, such as one typically provided by an operating system vendor (or by a network interface vendor supporting the network interface), is bound to the intermediary layer **102** in a conventional manner. The intermediary layer **102** is bound to the virtual NIC driver **204** instead of drivers **104, 106, 108** as depicted in FIG. 1. From the perspective of protocol stack **100**, the protocol stack is bound to a valid network interface.

The virtual driver **204** routes networking requests to the virtual protocol stack **202** which then repackages the network traffic for the different NIC drivers **104, 106, 108**. It will be appreciated that in accord with typical networking practices, return data will follow an inverse data path, allowing decryption of encrypted return data before the decrypted payload is given to the protocol stack **100**. However, before routing the networking traffic to NIC drivers **104, 106, 108**, the virtual driver **204**, the driver may make use of original driver capabilities (e.g., ability to ask a network interface to encrypt data) by way of communication links **206, 208, 210**.

Assume, for example, that NIC **116** has an on-board encryption ASIC, but NIC **218** and NIC **3120** do not. As will be discussed in more detail below, in such a circumstance, encryption for NIC **218** and NIC **3120** can be supported by routing encryption requests through NIC **116** encryption hardware and then repackaging the resultant encrypted data for delivery to NIC **218** and/or NIC **3120** by way of the virtual protocol stack **202**. That is, in one embodiment, network traffic to be encrypted would go from protocol stack **100**, to the intermediary **102**, to the virtual driver **204**, which communicates with the NIC **1** driver **104**

to have NIC **116** perform the encryption. The encrypted data is received by the virtual driver **204**, given to the virtual protocol stack **202**, which then re-sends the data for transmission by NIC **218** or NIC **3120**.

FIG. 4, is a flowchart illustrating using network interfaces to provide missing features, e.g., encryption, for other network interfaces, so as to provide a team of network interfaces apparently capable of homogeneously performing a function even though some of the network interfaces in fact cannot perform the function.

Assume the team is performing adapter fault tolerance (AFT) or adaptive load balancing (ALB), such as provided by the Intel Advanced Networking Services (iANS), and that the team is to be presented as capable of homogeneously providing IPSEC encryption support even though one or more members of the team does not have encryption support.

The phrase "Adapter Fault Tolerance" means presenting, to protocol stacks, several network interfaces (working as a team) as one network interface. One of these network interfaces acts as an active, or primary, network interface for network communication. When a fault in one of the underlying network interfaces of the team is detected, iANS switches the faulty member network interface with another member network interface known to be functional. Using AFT, network communication will be resilient to failure in the member network interface in use when fail-over to another functional member network interface occurs.

The phrase "Adaptive Load Balancing" means presenting, to protocol stacks, several network interfaces (working as a team) as one network interface, using all of the network interfaces as an active network interface for network communication. Outbound network traffic (transmit) is balanced between all team members comprising a fat channel capable to deliver high bandwidth. When a fault in one of the underlying network interfaces of the team is detected, iANS does not use the adapter, providing opportunity to replace the interface.

Note that IPSLC, AFT, and ALB are presented for exemplary purposes only, and that other encryption standards and networking capabilities are also intended to be supported as discussed herein.

In one embodiment, at least one of the network interfaces is based on an Intel 82559 or similar chipset providing IPSEC encryption support for a primary and a secondary use of the adapter. Primary use corresponds to use of a network interface to transmit and receive its own network traffic. Secondary use corresponds to use of a network interface to process data for an external entity, e.g., driver software for a different network interface, operating system component, API, or the like.

In secondary use, a network interface receives data from a requestor to be encrypted or decrypted. In one embodiment, the received data is processed and returned to the requestor. In another embodiment, the processing adapter processes and then directly transmits the data to the network for the requestor. For example, timing, throughput, or other considerations, may make direct transmission more efficient than returning the data for subsequent transmission. In one embodiment, the processing adapter is instructed to temporarily change its MAC address to the MAC address of the requestor's network interface lacking encryption support, so that responses to the transmitted network data will be received by the requestor's networking interface. Accordingly, network interfaces without IPSEC support may nonetheless process IPSEC network traffic by having the encryption processing handled by an IPSEC capable device.

5

The data to be secondarily processed can be stored in a host memory, such as in a main memory for a computing device housing the network interface, copied to a memory of the network interface, or stored in some other memory and made available to the network interface. It is assumed that Direct Memory Access, private or public bus, or some other communication pathway is available for receiving and returning data. Secondary use is intended to replace software encryption functions and consequently offload work from a host processor. When network interfaces having encryption support are present within a computing device, software encryption libraries can forward encryption tasks to the interfaces to be secondarily processed by the encryption hardware, interleaved with regular network traffic that goes out to the network.

Thus, to augment adaptive load balancing, adapter fault tolerance, or other networking functionality, a first operation is to identify 300 network interfaces bound to the augmentation layer 250 support IPSEC (or other functionality) to be shared. In one embodiment, the identification 300 operation confirms network interface identity data, such as vendor information and/or revision version of the network interface, to ensure compatibility with the augmentation layer. In a further embodiment, the augmentation layer refuses to operate with network interfaces not having particular identity data. For example, in such configurations, the augmentation layer may choose to only operate with network interfaces provided by the developer of the augmentation layer software and/or drivers.

A second operation is to verify 302 that at least one IPSEC capable interfaces provides secondary-use access to its encryption hardware. A single, fast, encryption component to an adapter may support encryption requirements for many other hardware devices. Alternatively, as discussed for FIG. 5, if multiple encryption-capable adapters are present, then all adapters can share task processing, e.g., operating as parallel processors.

If verification fails, then an adapter team cannot be heterogeneous shared, and sharing terminates 304. If verification succeeds, then the augmentation layer presents 306 itself to a protocol stack (e.g., protocol stack 100) as a network interface supporting IPSEC (or other desired functionality) with support for secondary use of its encryption hardware. Additionally, the augmentation layer may announce itself to an operating system as supporting secondary-use encryption tasks, thus allowing operating system APIs (e.g., Microsoft Windows CryptoAPI) to utilize encryption capabilities of the network interfaces.

The protocol stack then delivers 308 packets for transmission to the network 122 in either plain mode or encrypt mode. If 310 plain packets are to be sent, then the packets can be presented to an appropriate network interface's driver for transmission 312 in a customary manner. (Or they can be routed through the augmentation layer without any augmentation.)

However, if the packets are to be encrypted, then for each adapter that is to receive data for transmission, a check 314 is made to determine whether the adapter supports IPSEC transmissions. Note that depending on how one tracks which adapters can perform IPSEC transmissions, this check may or may not be literally performed. For example, a transmission mask may be employed to control which adapters simply send traffic without further review. It will be appreciated that which adapters receive data depends on transmission mode; thus, for example, under load balancing, all adapters receive a distributed portion of network traffic for transmission.

6

If the destination adapter does not support IPSEC, then the data payload for the destination adapter is sent 316 to a backup adapter that does support IPSEC. The backup adapter receives the data payload, encrypts 318 it pursuant to IPSEC, and returns 320 the encrypted data for delivery by the destination adapter as regular data. This arrangement allows load balancing (or other teaming algorithms) of IPSEC or other network traffic across a non-heterogeneous adapter team.

FIG. 5 is a flowchart illustrating one embodiment for processing receipt of network traffic sent according to FIG. 4. Generally, on receipt 322 of incoming network traffic, an inverse to FIG. 4 series of operations is performed. For example, assuming a networking mode of transmitting load balanced IPSEC traffic, if 324 an encrypted packet is received from a network, and if 326 received by a network interface which is IPSEC capable, then the received traffic will automatically be decrypted 328 by the adapter and presented 330 to the augmentation layer as a plain text packet. However, if the adapter is not IPSEC capable, then encrypted packets received by the adapter will be presented 332 to the augmentation layer still in encrypted form as received from the network.

The augmentation layer identifies 334 the encrypted packets as being encrypted, and forwards 336 them for decryption (e.g., as a secondary task) by an available IPSEC-capable adapter. Decrypted packets are received 338 and forwarded 340 by the augmentation layer in accord with a current processing algorithm, e.g., traditional (direct), fault tolerant, load balancing, etc., for presentation as regular plain text packets for processing by upper layer protocol stacks.

FIG. 6 illustrates an algorithm for using a team of network interfaces, controlled by an augmentation layer 250, to boost secondary use encryption by distributing an encryption task across multiple team members: in one embodiment, the proportional distribution of the task is according to a current workload of each network interface of the team.

Secondary use encryption throughput is therefore scaled according to the number of members in the team and their availability. Secondary use in adaptive load balancing mode can be performed by distributing encryption tasks to team members according to their current workload. Secondary use in adapter fault tolerance mode favors distributing encryption tasks to network interface team members which are inactive and waiting on failure of a primary running network interface. Such idle network interfaces can be used as dedicated encryption devices.

Note that spreading processing of encryption using load balancing techniques is not limited only to using network interfaces as hardware accelerators, but also to using other hardware devices which are capable of performing encryption, such as other encryption-capable devices within a computing device hosting the network interfaces. Additionally, note that load balancing and fault tolerance are used as exemplary operations that respectively utilize all network interfaces, or a single interface of a team. It is contemplated that the present invention will be applied to other tasks.

Operations 350-358 correspond to operations 300-308 of FIG. 4, and are only briefly discussed for FIG. 5. Thus, a first operation is to identify 350 network interfaces bound to the augmentation layer 250 support IPSEC (or other functionality) to be shared, and a second operation is to verify 352 that multiple IPSEC capable interfaces provides secondary-use access to its encryption hardware. If verification fails, then encryption processing cannot be spread across the identified adapters, and spreading terminates 354. If veri-

7

ation succeeds, then the augmentation layer presents 356 itself to a protocol stack as a network interface supporting IPSEC with support for secondary use of its encryption hardware, and may announce itself to an operating system.

The protocol stack then delivers 358 packets for transmission to the network 122 in either plain mode, encrypt mode, or for secondary processing with loop back to a requester (e.g., a protocol stack, encryption library or service, operating system, etc.). If 360 a network interface team is operating in adaptive load balance mode, the augmentation layer load balances 362 network traffic according to the network interface team's mode of operation.

If 364 a network interface team is operating in adapter fault tolerance mode, and if 366 regular network traffic, plain or encrypted is to be delivered to the primary (e.g., active) network interface, then the packets are delivered 368 to the primary adapter and transmitted to the network in a customary fashion.

If, however, non-regular traffic is received, e.g., secondary use data packets, then these packets are delivered to the backup network interface members such that they are balanced 372 across all available unused team members. If 370 the primary network interface has available resources, however, to process encryption tasks, then the primary adapter interleaves 374 secondary task processing with its primary transmission and receipt of network traffic. Remaining task processing is balanced 372 across all available unused team members. It is expected that appropriate queuing strategies will be employed to keep all adapters busy.

On receipt of network traffic, if the network interface team is operating in adaptive load balancing mode, or some other mode utilizing all network interfaces in the team, then if regular network traffic (plain or encrypted) is received, then it will be balanced across all team members as normal. If non-regular traffic is received, e.g., secondary use data packets, these packets are delivered to the all members of the network interface team such that they are balanced across all available team members.

Note that since encryption duties are separate from network transmission and reception, even if a network interface is defective or otherwise unable to process network transmissions, the network interface may still be functionally available for processing secondary use data. In one embodiment, when there are network interfaces that are not processing (or can not process) regular network traffic, these adapters will be first loaded with secondary use tasks to leave fully functional network interfaces available for processing regular network traffic. In addition, although not shown in these figures, processing accounts for the hot-swap removal and replacement of network interfaces. For example, if a defective network interface is replaced with a fully functional one, then the replacement interface should no longer receive a disproportionate amount of secondary use processing requests.

FIG. 7 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which portions of the invention may be implemented. An exemplary system for implementing the invention includes a computing device 400 having system bus 402 for coupling together various components within the computing device. The system bus may be any of several types of bus structures, such as PCI, AGP, VESA, etc. Typically, attached to the bus 402 are processors 404 such as Intel Pentium® processors, programmable gate arrays, etc., a memory 406 (e.g., RAM, ROM, NVRAM), computing-device readable storage-media 408, a video interface 410, input/output interface ports 412, and a network interface. A

8

mode 414 may provide an input and/or output data pathway, such as for user input/output, and may operate as a network interface in lieu of or in conjunction with other network interfaces 416.

The computing-device readable storage-media 408 includes all computing device readable media, and can provide storage of programs, data, and other instructions for the computing device 400 and components communicatively coupled thereto (e.g., a network interface card attached to the system bus 402). Media 408 includes hard-drives, floppy-disks, optical storage, magnetic cassettes, tapes, flash memory cards, memory sticks, digital video disks, and the like.

The exemplary computing device 400 can store and execute a number of program modules within the memory 406, and computing-device readable storage-media 408. The executable instructions may be presented in terms of algorithms and/or symbolic representations of operations on data bits within a computer memory, as such representation is commonly used by those skilled in data processing arts to most effectively convey the substance of their work to others skilled in the art. Here, and generally, an algorithm is conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities, and can take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. Appropriate physical quantities of these signals are commonly referred to as bits, values, elements, symbols, characters, terms, numbers, or the like.

The invention may therefore be described by reference to different high-level program constructs and/or low-level hardware contexts, and may be part of single or multiprocessor computing devices, such as personal computers, workstations, servers, etc., as well as hand-held devices and controllable consumer devices such as Personal Digital Assistants (PDAs), cellular telephones, or Internet television adapters. It will be appreciated that the invention can have its own processors, such as the Intel 82559 chipset providing IPSEC encryption support for network interfaces, and that these processors may operate asynchronously to, and possibly in conjunction with, host processors.

The computing device 400 is expected to operate in a networked environment 416 using logical connections to one or more remote computing devices 418, 420. In addition, the invention itself may operate in a distributed fashion across a network, where input and output, including user input and output (e.g., a graphical interface) may each occur at different networked locations. Thus, for example, assuming a perspective where computing device 400 utilizes a team of load balancing network interfaces, then remote computing devices 418, 420 include routers, a peer devices, a web server or other program utilizing networking protocols such as TCP/IP, IPSEC, IPX, hypertext transport protocol (HTTP), File Transfer Protocol (FTP), Gopher, Wide Area Information Server (WAIS), or the like.

It is understood that remote computing devices 418, 420 can be configured like computing device 400, and therefore may include many or all of the elements discussed for computing device 400. It should also be appreciated that computing devices 400, 418, 420 may be embodied as a single devices, or as a combination of separate devices; for example, a team of network interfaces may reside in a separate enclosure and be communicatively coupled to computing device 400 (e.g., by input/output interface ports 412 or other communication medium).

9

Having described and illustrated the principles of the invention with reference to illustrated embodiments, it will be recognized that the illustrated embodiments can be modified in arrangement and detail without departing from such principles. For example, while the foregoing description focused, for expository convenience, on using encryption hardware present in network interfaces to emulate encryption in non-capable network interfaces, and on distributing encryption tasks among multiple network interfaces, it will be recognized that the same techniques and analyses discussed above can be applied to other protocols and services. In particular, the encryption support need not reside in network interfaces, and instead may be provided by other components within a computing device.

And, even though the foregoing discussion has focused on particular embodiments, it is understood that other configurations are contemplated. In particular, even though the expressions "in one embodiment" or "in another embodiment" are used herein, these phrases are meant to generally reference embodiment possibilities, and are not intended to limit the invention to those particular embodiment configurations. These terms may reference the same or different embodiments, and unless indicated otherwise, are combinable into aggregate embodiments. Consequently, in view of the wide variety of permutations to the above-described embodiments, the detailed description is intended to be illustrative only, and should not be taken as limiting the scope of the invention. Rather, what is claimed as the invention, is all such modifications as may come within the scope and spirit of the following claims and equivalents thereto.

What is claimed is:

1. A method for sharing processing capabilities of utilizing multiple network interfaces among said network interfaces, comprising:

receiving a first network data to be transmitted by a first network interface according to a protocol;
determining the first network interface lacks hardware supporting the protocol;
providing said first network data to a second network interface different from the first network interface, the second network interface including hardware supporting the protocol;
transparently processing said first network data by the second network interface into a second network data according to the protocol; and
transmitting said second network data with said first network interface.

2. The method of claim 1, wherein the first network interface does not support the protocol, the method further comprising:

presenting said first and second network interfaces to a protocol stack as being a homogeneous team of network interfaces.

3. The method of claim 1, wherein the protocol includes encrypting the first network data before submitting said first network data to a network.

4. The method of claim 1, further comprising:
communicatively coupling a hardware-based encryption processor with said second network interface, said encryption processor performing said processing of said first network data.

5. The method of claim 4, wherein the hardware-based encryption processor supports a primary mode for encrypting network data for said second network interface, and a secondary mode for encrypting network data for said first network interface.

10

6. The method of claim 5, wherein the said first and second network interfaces operate in an adaptive load balancing mode, and wherein said second network interface interleaves said primary mode encryption with said secondary mode encryption.

7. The method of claim 6, further comprising:
providing a third network interface supporting the protocol;

wherein processing said first network data into said second network data is balanced across said second and third network interfaces.

8. The method of claim 7, wherein said balancing is performed according to a workload of said second and third network interfaces.

9. The method of claim 5, wherein the said first and second network interfaces operate in an adapter fault tolerance mode, and wherein said first network interface is a primary network interface, and said second network interface is a backup network interface.

10. The method of claim 1, wherein the said first and second network interfaces operate in an adaptive load balancing mode, and wherein said second network interface interleaves processing network data for said second network interface with processing said first network data into said second network data.

11. The method of claim 1, wherein the said first and second network interfaces operate in an adapter fault tolerance mode, and wherein said first network interface is a primary network interface, and said second network interface is a backup network interface.

12. A readable medium having encoded thereon instructions for sharing processing capabilities of multiple network interfaces among said network interfaces, the instructions capable of directing a processor to:

receive a first network data to be transmitted by a first network interface according to a protocol;
determine the first network interface lacks hardware supporting the protocol;
provide said first network data to a second network interface different from the first network interface, the second network interface including hardware supporting the protocol;
transparently process said first network data by the second network interface into a second network data according to the protocol; and
transmit said second network data with said first second network interface.

13. The medium of claim 12, wherein the protocol includes encrypting the first network data before submitting said first network data to a network.

14. The medium of claim 12, said instructions including further instructions to direct said processor to:

process said first network data into said second network data with a hardware-based encryption processor communicatively coupled with said second network interface.

15. The medium of claim 14, wherein the hardware-based encryption processor supports a primary mode and a secondary mode, said instructions including further instructions to direct said processor to:

encrypt network data for said second network interface when said encryption processor is in said primary mode; and
encrypt network data for said first network interface when said encryption processor is in said secondary mode.

16. The medium of claim 15, wherein said first and second network interfaces operate in an adaptive load balancing

11

mode, and wherein said second network interface inter-
leaves said primary mode encryption with said secondary
mode encryption.

17. The method of claim 16, in which a third network
interface supports the protocol, said instructions including
further instructions to direct said processor to:

balance processing said first network data into said second
network data across said second and third network
interfaces.

18. The medium of claim 17, wherein said balancing is
performed according to a workload of said second and third
network interfaces.

19. The medium of claim 15, wherein said first and second
network interfaces operate in an adapter fault tolerance
mode.

20. In a computing device, a network interface team,
comprising:

a first network interface lacking hardware support for a
protocol; and

a second network interface different from the first network
interface, the second network interface including hard-
ware supporting the protocol, said second network
interface configured to transparently process network
data for the first network interface if said network data
is to be transmitted according to the protocol and to
return processed data to the first network interface.

21. The network interface team of claim 20, further
comprising:

a first receiver, communicatively coupled to said first
network interface, for receiving network data to be
transmitted by said first network interface;

a second receiver, communicatively coupled to said sec-
ond network interface, for receiving network data to be
transmitted by said second network interface; and

a transferor, communicatively coupled with said first
network interface and said second receiver, and con-
figured to transfer network data to said second network
interface for processing according to the protocol.

22. A method for sharing processing capabilities of mem-
bers of a system of network interfaces communicatively
coupled with and operable to communicate over a network,
comprising:

determining a first network interface is to transmit first
data having a data configuration;

determining the first data is configured in accordance with
a protocol unsupported by the first network interface;
locating a second network interface of the system includ-
ing hardware that supports the data configuration;

12

transparently secondarily processing by the hardware of
the second network interface of the first data in accor-
dance with the protocol into a second data; and

providing the second data to the first network interface so
that the second data appears to have been processed by
the first network interface.

23. The method of claim 22, further comprising:

selecting the first network interface to transmit the first
data based at least in part on a load-balancing of
network traffic across the plural network interfaces;

performing by a driver for the first network interface of
said determining the first data is configured according
to the protocol unsupported by the first network inter-
face;

receiving by the driver of the second data, wherein the
data is now in a format supported by the network
interface; and

providing by the driver of the second data to the first
network interface.

24. A method for distributing network processing across
a team of network interfaces cards including at least a first
network interface card (NIC) lacking support for a first
specialized capability and a second NIC that supports the
first specialized capability, the method comprising:

receiving first data to be processed and transmitted by the
first NIC to a recipient;

determining processing said received first data requires
the first specialized capability unsupported by the first
NIC;

transparently secondarily processing by the second NIC
of the first data into second data with the supported first
specialized capability; and

providing the second data to the first NIC for transmission
by the first NIC to the recipient.

25. The method of claim 24, wherein the second NIC
comprises an application specific integrated circuit provid-
ing the first specialized capability.

26. The method of claim 24, wherein the team of network
interfaces include a third network that supports a second
specialized capability, the method comprising:

aggregating specialized capabilities offered by interfaces
of the team; and

providing a virtual NIC appearing to provide each of the
specialized processing capabilities.

* * * * *

LAYER 4+ SWITCHING WITH QOS SUPPORT FOR RTP AND HTTP

Till Harbaum, Martina Zitterbart
 Institute of Operating Systems and Computer Networks
 Technical University of Braunschweig
 Bültenweg 74, D38106 Braunschweig, Germany

Frederic Griffoul, Jürgen Röthig, Sibylle Schaller, Heinrich J. Stüttgen
 Computer and Communication Research Laboratories
 NEC Europe Ltd.
 Adenauerplatz 6, D-69115 Heidelberg, Germany

Abstract

This paper provides an overview over current approaches and applications of layer 4 switching (L4Sw) and outlines a scheme for QoS support based on layer 4 and higher layer information. Today, L4Sw is mainly used for filtering in the context of firewalls. Additionally L4Sw has the potential of introducing per flow QoS support without the need for complex out-of-band signaling. We used traffic measurements on a real router to determine which higher layer protocols generate the most traffic and should therefore be investigated first. From the measurements it is obvious, that HTTP is the most important candidate. Experiments on extracting TCP and HTTP information demonstrated the feasibility of Layer4+QoS support with respect to processing power and storage capacity demands of intermediate systems. In addition to the TCP-based HTTP protocol, future investigations will be done for RTP as a key protocol for multimedia traffic.

Introduction

Traditionally, switching is a cell, frame or packet based network interconnection technique operating at layer 2 of the OSI reference model (e.g. ATM, Frame Relay and the like). This is in contrast to routing, which interconnects subnetworks at layer 3 of the OSI model. Generally speaking, a lower layer network interconnection (i.e. switching) performs more efficient than a higher layer interconnection (i.e. routing), whereas a higher layer interconnection provides better control and flexibility than a lower layer interconnection. In order to combine the benefits of both approaches, a new family of "integrated switch routers (ISRs)" a.k.a. IP switches has evolved over the last couple of years. However, recently there is an increasing demand for improved network services with regard to security, performance or QoS support for multimedia traffic. New switching techniques, using higher layer (Layer 4 and above) control information are being considered to provide these improvements. One of the main ideas behind this so called L4 switches (L4Sw) is, that higher layer information within the first packet(s) of a data flow can be utilized to set up a switched path for this flow. If this scheme can be extended to first detect and then to allocate the appropriate QoS for such a flow, we will have an attractive way of handling QoS allocation in-band, without

explicit and complex out-of-band signaling. However, this approach puts additional burden on the data forwarding path, which is highly performance critical. It is the main objective of this study to investigate, whether such a signaling-less QoS detection scheme is feasible and practical, i.e. whether the advantages of this scheme outweigh its disadvantages.

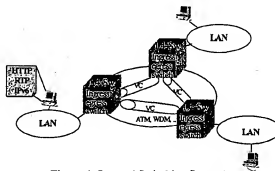


Figure 1: Layer 4 Switching Scenario

L4Sw is an approach that could be beneficially used at the edge node of a multi-gigabit IP backbone, independent of whether the backbone is based on ATM or other high performance transport technologies like WDM. High priority data or real-time flows may be detected as they leave the high-performance, typically over-provisioned LAN and enter the IP backbone, which generally requires explicit QoS support for performance and/or charging reasons.

A flow is defined as a number of related packets sent by applications over the network. Each flow has certain features. For instance, the download of content from a Web server which may be a collection of different data flowing between the server and a host, or the transmission of video or audio using special codes.

Within this report, the use of layer 4 and above information to implement QoS support in IP backbones, is analyzed in detail. We briefly review the current state-of-the art in L4 switching and then identify the main technical challenges in the design of a QoS supporting L4 switching architecture, including an analysis of the implementation complexity of such an approach. Third we show traffic meas-

urements on an existing WAN ingress router, to identify the traffic flows that will actually be able to benefit from such an architecture.

Layer 4 Switching

Layer 4 switching uses transport layer information, e.g. TCP or UDP port numbers, to forward packets. Using layer 4 information to forward data allows to collect additional information about source and/or destination of a data frame. Not only the destination machine, but also the type of program transmitting and receiving the data units via well-known ports can in many cases be determined by the layer 4 information. Today's routers already use this information to enable or disable selected services (firewall).

Layer 4+ Switching

Layer 4+ switching (L4+Sw) describes the ability of a switch to look inside higher layer protocol information within a packet; i.e. L4+Sw accesses application header information for protocols built on top of TCP/UDP, like the HTTP or the RTP/TCP headers.

In addition to layer 3 address information used for IP routing and layer 3 switching, layer 4+ switching uses information from the transport layer and above to forward data. The transport layer includes TCP and UDP and controls end-to-end communication between applications.

At layer 4 each TCP stream or UDP packet is associated with two port numbers. These port numbers identify the application protocol (FTP, HTTP, Telnet...) used with the corresponding communication. The protocol number is used at the IP layer to determine the appropriate transport protocol that has to handle the packet.

Some of these TCP/UDP port numbers have a determined meaning in all TCP/IP implementations [RFC1700]. The additional information supplied by the TCP/UDP port numbers is used for layer 4 switching.

Some protocols like RTP do not use well-known ports, other protocols like HTTP are not forced to always use their well-known ports. In these cases the port number itself does not help to determine the flow type. Additional information from layers above layer 4 (L4+) is needed to determine the flow type.

Applications and state-of-the-art

Several different approaches of L4Sw are currently used or discussed. They differ in the information extracted from the higher levels and the usage of this information. Today's main applications are:

Firewalls use layer 4 information inside the router for filtering. A firewall decides on the layer 4 port numbers whether information is forwarded or discarded for security reasons. Firewalls are easy to implement and consume only few additional CPU cycles inside a router to look up the

port number in the packet. This does not affect the routing tables of the router and, thus, can easily be added into existing routers. A firewall approach is discussed in [YAGO98].

QoS support based on layer 4 information uses the well-known port addresses to assign different service classes (priorities) to different data streams. To support QoS with layer 4+ information, the router has to extract content type information and the like from the data passing by in real-time. Classification based on well-known port numbers gives very coarse results. It fails completely for protocols not using well-known ports, e.g. RTP. For a fine grained classification further information from layer 4 and above is needed. This information can be used to determine the contents of for instance HTTP connections (text, graphics, audio, video...) and to detect and classify IP flows using protocols like RTP (audio, video, and whiteboard).

Current Layer 4 implementations

The existing implementations implement security filter and firewalls (YAGO, Foundry, Alton). QoS support based L4Sw is mentioned in [BrPa97, YAGO98, Foun98], but no implementation or details are available.

Technical Challenges of L4+Sw

Adding layer 4+ support to the existing router and switch concepts may tighten some already existing bottlenecks of the packet forwarding process. Performance loss due to even more complex search and update operations may be the result. Other problems may arise due to the fact that the switch now needs to access the encapsulated data where classic routers and switches just don't need to care about the payload field and its encoding. The major technical challenge will presumably be the search engine using string comparisons to filter out HTTP parameters. Furthermore, the mapping of layer 4(+) based QoS parameters onto integrated/differentiated services has to be done by the switch. Another open challenge is the handling of encrypted data. Obviously encryption prevents the switch from interpreting the packet content. For a more detailed discussion see security section below.

Search Table

Depending on the kind of usage of layer 4+ information, the demands on CPU power and memory inside the switch can be very high [PeZu92]. Standard IP routers need to store information about all possible destinations [TakZ94, ZHMB97, HMZB98, DBCP97, NiKa98]. This information may be compressed by using a net mask. In high-end routers, routing tables store up to 64k entries and a powerful search engine and algorithm is needed. With gigabit routing these tables quickly become system bottlenecks. Therefore, specialized hardware solutions are being developed to support scalable, fast and efficient table lookup. The hardware presented in [HMZB98] allows concurrent

FPGA based search, which is especially suited for multiple search algorithms needed with layer 4 switching as presented in [SVSW98]. According to [KeSh98] the problem of building fast and cost efficient table lookup support is a solved problem today.

Support for layer 4 port number handling increases the sizes of these search/routing tables dramatically, since for every entry additional information has to be stored. Standard IP routers need to hold one entry per destination address. Since L4+Sw handles different protocols separately, a L4+Sw needs to store one entry for each protocol used on each machine. As routing table lookup has already been a bottleneck for standard IP routers, fast lookup will become an even more important issue with L4Sw (Table 1).

	Standard IP router	Layer 4 switch
Table length	Number of destinations	Number of applications
Entry length	Length of IP address	Length of source and destination IP + port + additional information
Increased search complexity due to:	N/A	Less compression, longer tables, longer keys ...

Table 1: Router demands for Layer 4 switching

Furthermore the length of the keys is increased by the source address (which is not used in IP routers, but which is needed to identify a TCP connection) and the port numbers of source and destination and additional information extracted from layer 4 information.

The layer 4+ analysis and routing algorithm must be capable of handling keys of up to three times the length of keys used in standard IP routers. Also, the search engine has to cope efficiently with tables that are significantly larger than the routing tables of today's layer 3 switches and routers. A fast hardware implementation is needed to fulfill these demands.

Mapping of Traffic Parameters

The information derived from layer 4 and above needs to be mapped to the QoS parameters or classes of the respective network QoS model used, like IP differentiated services, IP integrated services, ATM, or other future QoS architectures.

This requires the isolation of traffic parameters and the interpretation of implicit service requests and priorities, given by the information extracted from layer 4 and above. In addition signaling for explicit reservations may be included to allow applications to reserve dedicated resources for a stream.

Security

With regard to secure communications the following needs to be considered: Is the data to be transmitted sensitive or confidential? Whom can I trust, and how much trust is appropriate? Depending on the answer to these and similar questions appropriate measures must be taken to protect the data during the transmission.

One such measure is encryption. But, encryption poses a problem to Layer4+ flow detection, because it may be impossible to access the necessary information without decrypting all or a certain part of the IP payload. However, it won't be done since CPU consumption and complexity are too high.

There are several options for Layer4+ flow detection in connection with security:

1. Don't use encryption.

This option, although best for Layer4+ flow detection and switching, is not acceptable in a lot of cases.

2. Encrypt at the ingress node into a L4+Sw cloud.

This allows for performing Layer4+ switching tasks and provides security over the switched path. This is viable only if the ingress node is inside the trusted part of the delivery path between the sender node and the first untrusted section towards the receiver. The algorithms and keys for encrypting packets must be known at the L4+Sw ingress node, which is feasible if the L4+Sw ingress node is inside the trusted region, e.g., located on the corporate premises. Also, the necessary resources for this additional task must be available (see Figure 2, option 2).

3. Explicit reservations for encrypted data streams.

A sender who wants to send encrypted data streams requests explicit reservations for a switched path towards the sender with the help of a reservation protocol, e.g. RSVP or ATM. This option does not use information derived from L4+Sw and may be used in combination with layer 4 based QoS.

4. Use of explicit flow identifiers like IPv6 Flow Labels

Here the data is encrypted at the sender. Packet classification is based on the Flow Label field of the IPv6 protocol (see Figure 2, option 3+4).

For option 1, no special effort is needed for Layer4+ switching. The interesting options are the options 2 and 4. When choosing option 2, encryption must be applied to the data after the flow detection test was done. For option 4 the first packet of a flow must be unencrypted, so that the QoS detection can work properly. All further packets of a flow may then be encrypted but can be classified properly based on the flow label.

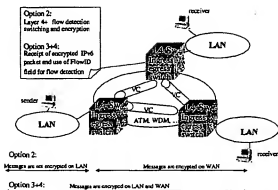


Figure 2: Encryption/Decryption on the Ingress Switch

TCP Traffic Measurements

To gain information about the distribution of different flow types to be expected, measurements on a router of the Technical University of Braunschweig have been performed. This router is in fact the ingress router from the institute network to the wide-area and the TU Braunschweig campus network, and therefore matches well the environment we are focusing on.

Basic scenario

The router used for the feasibility tests connects the Institute of Operating Systems and Computer Networks at the Technical University of Braunschweig with the internal campus net and the external Internet.

The local institute network consists of approximately 80 computers. These include local WWW and FTP servers, allowing us to monitor incoming HTTP and FTP requests. On the other hand, the internal network includes a workstation cluster for student lessons, generating a considerable amount of outgoing HTTP, FTP and other requests.

The router is running Linux, allowing modifications of the router software and direct access to the router network interfaces to capture the raw net traffic. To get this direct access, the network interface hardware was switched into promiscuous mode.

On the router a background process was used to monitor the passing traffic and to verify the L4+Sw algorithms. This process bypassed the TCP/IP processing within the Linux kernel. However, for the actual L4+ switching prototype, a kernel implementation has been developed for performance reasons.

Traffic monitoring

The first step of analysis was to take a look at the whole TCP traffic with respect to the port numbers in the TCP header.

Figure 3 shows the traffic monitoring over one week. It clearly shows, that most of the traffic is generated by HTTP and FTP. Other services using well-known ports like SMTP (mail), DNS (name resolution), X11 (X window system), SSH (secure shell) and NNTP (net news) generate little traffic. Some traffic is transferred over ports not linked to specific protocols or applications. This traffic includes for example students playing games over the net and experimental traffic, but it also contains HTTP traffic, that is not using the well known port for HTTP traffic (port 80). The results regarding HTTP and FTP match the results in [AMiz99, ThMW97]. These papers show an even higher amount of HTTP traffic.

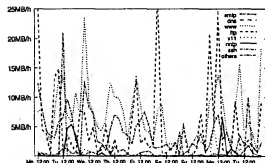


Figure 3: TCP/IP traffic monitored

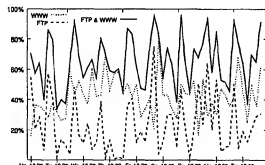


Figure 4: One week of FTP/WWW router traffic

While Figure 3 displays the absolute traffic in Megabytes per hour, Figure 4 shows the relative amount of FTP and WWW traffic compared to the overall TCP traffic. This figure clearly shows, that at least 40% of the overall traffic is either HTTP or FTP based. At most 93% was FTP or HTTP traffic and on average 70% was HTTP or FTP.

Due to the wide-spread use of the web for consumer applications, there will probably be an even higher amount of HTTP traffic on commercial routers operated by ISPs than the measurements from our university network show.

The TCP segmentation problem

A more detailed HTTP traffic analysis requires search engines to access the payload field inside the TCP transfer units, since HTTP is a TCP based protocol. Thus, TCP segmentation and reassembly inside the router will take place. Figure 5 shows an example of a TCP connection. In this example, the string 'This_is_a demonstration' is transferred over a TCP/IP connection. Due to the internal TCP segmentation the string may be transmitted in small portions. In the example, the string is split into four TCP segments. If a search unit inside a router tries to find the string 'demonstration' in the passing traffic, it will never find the message, since there is no single packet containing the word 'demonstration'.

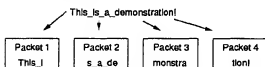


Figure 5: TCP segmentation

Therefore, for 100% error free HTTP analysis segmentation and reassembly has to be done for every stream to be analyzed inside the router. This means complex operations and up to 64k (the window size of TCP) buffer required for every TCP stream.

To get real life values on segmentation and reassembly, a TCP traffic analyzer was implemented on the router. This analyzer ran for about 30 hours. During this time all kinds of long and short distance connections were detected. There were low bandwidth connections as well as connections with a huge throughput and the internal servers were accessed from the outside as well.

The main results of these measurements are:

- The smallest single TCP segment contained 256 bytes payload. This segment size is big enough to hold the information needed for HTTP identification.
- All traffic was HTTP 1.0 based. The Results may vary with future HTTP 1.1 usage, because HTTP 1.1 allows aggregation of multiple HTTP transfers into one TCP connection.
- Over 99% of all HTTP headers could be identified as HTTP traffic by looking at one single TCP segment.

The most important result from this analysis is the fact that without performing a TCP reassembly inside the router over 99% of all HTTP traffic could be classified correctly. This allows fast and efficient HTTP/TCP processing inside the router without the demand of reassembly units and huge reassembly buffers and the additional delay.

HTTP traffic analysis

To get further information on HTTP handling, a HTTP analysis was started on the router. For layer4+ switching additional information from the HTTP header is used for classification and routing decisions.

HTTP header information of interest for layer 4+ switching may for example be:

- The name of the file being transmitted
- The type of the data transmitted (video, audio, text...)
- Source and destination of the data transmitted
- The expected transmission length

All this information may be included in the HTTP header, but most of this is optional. The measurements show that lots of connections don't use the Content Type field. Further investigation showed that the HTTP streams without valid Content Type could be identified by:

- The file name (.gif, .jpg, .html) contained in the preceding GET request from the client.
- If this file name was of unknown type, the content was in all cases plain HTML text.

All other traffic, especially high bandwidth connections for video and audio transmissions, had a valid Content Type field in the header of the HTTP reply (e.g. 'MPEG video' for MPEG-1 video transmission or 'RealAudio' for RealTek audio transmission).

This way the content type of all detected HTTP flows (i.e. 99% of all HTTP flows) could be determined.

Another interesting header field is the Content Length field. This field allows knowing already at the beginning of the transmission how many data follows. The measurements showed that 77% of all HTTP connections had a valid Content Length field. These connections were carrying about 87% of the overall HTTP traffic. This means that the transmissions without Content Length field are shorter than the average HTTP transmission. Other results from these measurements are:

- A connection transferred an average of 7590 bytes.
- 32% of all connections are shorter than 1000 bytes.
- The average amount of data transferred between two end systems was about 32900 bytes (giving an average of 4.3 connections per client server pair)

These measurements were done on the connection between the router and the external Internet and they exclude internal traffic between the institute and the campus network. During monitoring of the university's internal traffic a dramatic increase of the average amount of data transferred between two end systems was observed. This means that local HTTP connections last longer and the clients are more rarely 'hopping' between different servers. The aver-

age amount of data between two end systems was bigger than 300 kilobytes per session. This is ten times the value of the connections to the external Internet.

Implications

From the measurements presented in the last section we conclude, that the CPU and memory intensive segmentation is not really needed here. In the worst case, some HTTP flows may be missed and get the same treatment as they would without L4+ switching. Therefore HTTP analysis consists of two main steps:

step 1: detection and skipping of IP and TCP headers

Since this step only requires to extract both header length fields and to skip the complete IP and TCP headers by increasing the pointer to the payload field, this can be done with very low effort. This step is easy to implement and can easily be done in hardware.

step 2: search for 'magic strings' in payload

This is a complex task, requiring the search engine to step through the payload field and apply string matching functions. This step can be implemented in hardware, but requires a more complex hardware, than step 1.

The first step requires only a low effort and can easily be implemented in hardware. The more time and processing power consuming step is the second one which requires up to 200 comparisons for every single TCP packet. But since this information is only needed while the status of the connection is unknown this is only needed for very few packets (in average <5% of all packets need to be analyzed). However, a powerful hardware unit may be useful to support IP/TCP header skipping and fast pattern search in the payload field.

Conclusion

Layer 4+ information extracted from HTTP and other protocols like e.g. RTP traffic can be used for various applications like flow aggregation, reservation, prioritization.

First it can be applied to do packet classification for Differentiated Services, i.e. to support the Class of Service model for multimedia traffic.

Alternatively HTTP based layer 4 information can be used for bandwidth reservation. One obvious application is to reserve bandwidth for audio and video connections and to automatically route all audio and video traffic from a particular source through these reserved connections. Another application is to reserve bandwidth for dedicated client-server pairs and protocols.

Further, in the case of HTTP traffic, flow aggregation may be used to bundle several flows into one ATM VC or similar. It may be useful to bundle a couple of low traffic

connections or to bundle connections of the same type (one VC for HTML, one for graphics, one for audio...) or just to have one channel for all HTTP connections between a client-server pair.

Information for traffic handling may be derived from Layer 4+ information as discussed in this report. Although our first prototype is implemented in software, this function eventually requires hardware support, especially when used on a gigabit backbone network as intended.

References

- [Alte98] Alteen Inc., Scaling Server Application Performance with Layer 4 Switching, TR March 1998.
- [AMlz99] J. Aracil, D. Morato, M. Izal, Analysis of Internet Services in IP over ATM Networks, Proc. Of the 2nd Int. Conf. On ATM, Colmar, June 1999, pp.258-266
- [BrPa97] J. Bransky and L. Passmore, Layer 4 Switching, White Paper, Technical Report, Sept. 1997.
- [DBCP97] M. Degermark, A. Brodnik, S. Carlsson and S. Pink, Small Forwarding Tables for Fast Routing Lookups, In Proceedings ACM SIGCOMM '97, Cannes, France, Sept. 1997.
- [DoKN96] W. Döringer, G. Karjoth and M. Nashehi, Routing in Longest-Matching Prefixes, In IEEE/ACM Transactions on Networking, volume 4, Sept. 1996.
- [Foun98] Foundry Inc. Cutting Trough Layer 4 Hipe. Technical report, Jan. 1998.
- [HMBZ98] T. Harbaum, D. Meier, M. Zitterbart, D. Brökelmann, Hardware Assist for IPv6 Routing Table Lookup, In SYBEN '98, Zurich, Switzerland, May 98.
- [NiKa98] S. Nilsson and G. Karlsson, Fast Address Lookup for Internet Routers, In Proc. IFIP/Broadband Communications, University of Stuttgart, April 1998.
- [PeZu92] T. Pei and C. Zukowski, Putting Routing Tables in Silicon, In IEEE Network Magazine, volume 6, IEEE, Jan. 1992.
- [RFC1700] J. Reynolds and J. Postel, ASSIGNED NUMBERS, RFC 1700, ISI, Oct. 1994.
- [SVSW98] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, Fast and Scalable Layer 4 Switching, July 1998.
- [TaKZ94] A. Tantawy, O. Koufopavlou and M. Zitterbart, On the Design of a Multigigabit Router, In Journal on High Speed Networks, volume 3, 1994.
- [ThMW97] K. Thompson, G. J. Miller and R. Wilder, Wide-Area Internet Traffic Patterns and Characteristics, IEEE network, vol. 11, no.6, December 1997
- [Yago98] YAGO Inc. Layer 4 Switching: An Overview. Technical report, March 1998.
- [ZHMW97] M. Zitterbart, T. Harbaum, D. Meier and D. Brökelmann, Efficient Routing Table Lookup for IPv6, IEEE HPSC '97 Workshop, Chalkidiki, Greece, 6/97

Optimizing TCP Forwarder Performance

Oliver Spatscheck, Jørgen S. Hansen, *Student Member, IEEE*, John H. Hartman, *Member, IEEE*, and Larry L. Peterson, *Senior Member, IEEE*

Abstract—A TCP forwarder is a network node that establishes and forwards data between a pair of TCP connections. An example of a TCP forwarder is a firewall that places a proxy between a TCP connection to an external host and a TCP connection to an internal host, controlling access to a resource on the internal host. Once the proxy approves the access, it simply forwards data from one connection to the other. We use the term *TCP forwarding* to describe indirect TCP communication via a proxy in general. This paper briefly characterizes the behavior of TCP forwarding, and illustrates the role TCP forwarding plays in common network services like firewalls and HTTP proxies. We then introduce an optimization technique, called *connection splicing*, that can be applied to a TCP forwarder, and report the results of a performance study designed to evaluate its impact. Connection splicing improves TCP forwarding performance by a factor of two to four, making it competitive with IP router performance on the same hardware.

Index Terms—Firewall, proxy, router, TCP.

I. INTRODUCTION

IT IS increasingly common that processes communicate with each other indirectly through a proxy. This happens, for example, in a firewall where a proxy mediates the flow of information between a TCP connection to an untrusted external entity and a TCP connection to a trusted local entity. We use the term *TCP forwarding* to denote the general pattern of indirect communication over a pair of TCP connections via a proxy.

One consequence of TCP forwarding is that there is often a single network node—e.g., a firewall—that runs proxies on behalf of many different indirect communications. This network node, which we call a *TCP forwarder*, plays a role very similar to that of an IP router, except it must execute two TCP endpoints and a proxy for every “flow” that passes through it. To intercept the TCP connections successfully it has to receive *all* TCP packets for both TCP connections. This can be achieved by either addressing the TCP forwarder directly or by placing it on a choke point in the network. Therefore, the performance of the TCP forwarder, i.e., its throughput in terms of packets-per-second, can play a significant role in the network performance perceived by the communicating entities.

Manuscript received May 15, 1998; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Pink. This work was supported in part by Defense Advanced Research Projects Agency Contract DABT63-95-C-0075 and Contract N66001-96-8518, and by National Science Foundation under Grant CCR-9415932.

O. Spatscheck is with AT&T Labs—Research, Florham Park, NJ 07932-0971 USA (e-mail: spatsch@research.att.com).

J. S. Hansen is with the Department of Computer Science, University of Copenhagen, Copenhagen, Denmark (e-mail: cyller@diku.dk).

J. H. Hartman is with the Department of Computer Science, University of Arizona, Tucson, AZ 85721 USA (e-mail: jhh@cs.arizona.edu).

L. L. Peterson is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA (e-mail: llp@cs.princeton.edu).

Publisher Item Identifier S 1063-6692(00)00322-7.

This paper makes three contributions. First, it defines briefly a general framework for TCP forwarding, and demonstrates the relevance of TCP forwarding to three applications: firewalls, HTTP proxies, and mobile computing. Second, it describes an optimization technique, called *connection splicing*, that can be used to improve the performance of a TCP forwarder. An implementation of connection splicing in the Scout operating system is also presented. Third, it reports the results of a performance study that measures the effectiveness of connection splicing. The study shows that connection splicing improves TCP forwarder performance by a factor of two to four, bringing its performance close to that of an IP router implemented on the same platform.

II. TCP FORWARDING

When two entities communicate indirectly through two separate TCP connections, an entity called a *proxy* mediates the communication, interposed between the two connections, and controls the flow of data between the communicating parties. The proxy decides if the parties can communicate, and if so, what is communicated. A proxy can both restrict and enhance the communication. For example, a Telnet proxy can restrict to which computers the outside world may connect, and perhaps which users may log in. On the other hand, a Telnet proxy could also serve as a clearinghouse for a collection of servers by providing a single connection point for outside Telnet accesses. The Telnet requests are processed by the proxy and forwarded to the appropriate computer, shielding the outside world from the internal structure of the site.

We use the term *TCP forwarding* to refer to communication relayed over two TCP connections via a proxy. TCP forwarding is not as simple as copying bytes from one connection to the other, however. The proxy must control the communication as well as relay bytes, and therefore, a proxy has two modes: *control mode* and *forwarding mode*. In control mode the proxy processes either out-of-band or in-band control information. Once the control functions have been completed, the proxy switches to forwarding mode to move data between the connections. After the data transfer, the proxy may switch back to control mode. For example, a Telnet proxy starts off in control mode, and processes a Telnet request to determine if the connection should be allowed, based on the target machine, port, and perhaps user ID. Once the connection has been completed, the proxy switches into forwarding mode to transfer data between the two computers. Switching between these two modes of operation is the primary difficulty in developing an optimized TCP forwarding mechanism.

The processing done in control mode varies greatly between proxies, ranging from very little processing during connection

setup, to continuous monitoring of the data stream while forwarding to extract control information. Proxies can be broadly classified into four categories, depending on the degree of control processing they do.

The first class of proxies perform a minimum of control processing; they typically perform level-4 routing based on IP addresses and port numbers. They are in control mode only during connection setup, after which they switch to forwarding mode for the duration of the connection. An FTP proxy is an example: it processes an FTP request in control mode on the control connection, sets up a data connection between the two computers, and switches to forwarding mode on the data connection until it is closed. The control connection remains in control mode to process subsequent FTP requests.

The second class of proxies performs more control processing because they authenticate the user or request and base routing decisions on either the result of the authentication or control information passed in the TCP connection. A Telnet proxy is a member of this class. Typically, a Telnet proxy requests a user ID, password, and the destination of the Telnet request. This information is received on the TCP connection by the proxy and is used to authenticate the user and establish a connection to the correct remote machine. At this point, the proxy simply forwards data between the two connections.

The third class of proxies remains in control mode for all data transferred in one direction, but switch to forwarding mode for data transferred in the other. An example is an HTTP proxy that processes the HTTP requests (control information) sent by clients, but simply forwards the data returned by the HTTP server.

The fourth class remains in control mode and continuously monitors data passed in both directions. This might be the case for a proxy that allows users on a protected network to access HTTP servers on the Internet. The proxy could filter outgoing accesses to restrict the servers that can be reached, and filter incoming access responses to remove (untrusted) Java code.

TCP forwarding has many uses, including such diverse functions as a network firewall, an HTTP proxy, and a mobile computing system. These three examples illustrate the power of TCP forwarding, and motivate the need for an efficient implementation.

A. Firewall

A firewall provides limited connectivity between a protected network and the relative chaos of the Internet, as shown in Fig. 1. The firewall contains different types of proxies, each handling a different type of communication between the two networks, such as Telnet, FTP, etc. A typical proxy accepts connections on one network, authenticates the entity making the connection request, and forwards the data to the other network, perhaps after applying a filter. The firewall either uses its own IP address (*classical proxy*) or is completely transparent to the user (*transparent proxy*) [6]. A classical proxy must use the control information in the request to determine the connection's true destination.

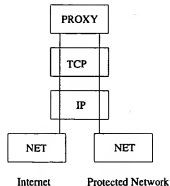


Fig. 1. Overview of an application-level firewall. Data from one network pass through the proxy which forwards them to the other network if the desired security guarantees are not violated.

B. HTTP Server Proxy

TCP forwarding can also be used to develop scalable servers such as HTTP servers. HTTP server names are embedded in the URL namespace, making it difficult to implement a single HTTP service from a collection of servers. Load-balancing across the collection is a problem; web sites typically offer the users a selection of servers from which to choose, manipulate the DNS mappings to change dynamically the IP address associated with a site name, or use the HTTP redirection mechanism to redirect requests to unloaded servers. The first two offer coarse-grained load balancing, while the last requires two HTTP connections per URL accessed.

An HTTP server proxy that forwards TCP connections is a better solution. Clients connect to the proxy, which processes their requests and forwards them to the appropriate server. The proxy must continually monitor the data received from the clients, however, so that requests can be extracted and processed, and the connections re-forwarded as appropriate. The data returned from the servers, however, is simply forwarded to the clients.

Such an HTTP proxy might implement a variety of forwarding policies, in addition to load-balancing over a set of homogeneous servers. The proxy could forward connections to servers based on the URL requested, allowing a collection of servers, each of which serves a different collection of pages, to appear as a single site. It could also provide more complex functionalities as described by Brooks *et al.* [5].

C. Mobile Computing

Our final example involving proxies is from the area of mobile computing. Here proxies are used to improve the performance of mobile hosts operating across wireless links by separating TCP connections into two connections; one covering the wireless link and one covering the wired network. The performance enhancement can either be simply an improvement caused by the separation of flow control on the two different types of network, or it can rely on transformation or filtering of data, e.g., the proxy reduces the resolution on graphics sent to the mobile host over a low capacity link and removes all video clips from e-mail. The situation is complicated by the fact that mobile hosts often use a mixture of wireless and wired networks, switching between them on the fly. When the mobile host is con-

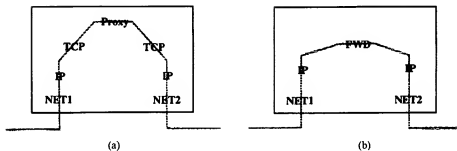


Fig. 2. Optimizing two TCP connections into a single spliced connection. (a) Unoptimized TCP forwarder. (b) Optimized TCP forwarder (with spliced connection).

nected to a wired network, the proxy merely relays data in the forwarding mode, but cannot be removed from the path of communication due to the presence of the bipartite TCP connections.

Another use of proxies is to allow a mobile host to change its point of attachment to the network without jeopardizing any open connections. In this case the proxy would operate in the forwarding mode when the mobile host is connected, but would switch to control mode both when the mobile host connects and when it disconnects. This would allow the mobile host to terminate its TCP connections, move to a new location with a new IP address, and establish a new set of TCP connections to the proxy without affecting the peer hosts on the other side of the proxy.

III. CONNECTION SPLICING

This section describes an optimization technique, called *connection splicing*, that improves TCP forwarding performance. It includes a discussion of the many complications that make connection splicing difficult in practice. To simplify the following discussion, we focus on the flow of data in a single direction; the same work must also be done for data going in the other direction.

A. Overview

The proxy involved in TCP forwarding operates in either control mode or forwarding mode. The basic idea of connection splicing is to detect when a proxy makes a transition from control mode to forwarding mode, and then splice the two TCP connections together into a single forwarding path through the system. The resulting spliced connection replaces the processing steps (and associated state) required by two TCP connections with a single reduced processing step (and associated state).

Fig. 2 schematically depicts the optimization. The standard (unoptimized) forwarder on the left requires TCP segments to traverse TCP twice, with each instance of TCP maintaining the full state of the connection. In this case, the proxy simply passes segments from one connection to the other when it is in forwarding mode. The optimized forwarder on the right replaces the proxy and two TCP processing steps with a single FWD processing step. FWD maintains just enough state to forward TCP segments successfully from one network to another. The state FWD needs to maintain is described later in this section.

A single proxy might require both configurations, however. The configuration on the left *must* exist when the proxy is in

control mode; the proxy must be in the loop because it needs to inspect the data flowing between the two TCP connections. The configuration on the right *may* exist while the proxy is in forwarding mode. Forwarding can also happen in the left configuration, but performance suffers. With this perspective in mind, there are three cases to consider: how the optimized configuration on the right works in the steady state (Section III-B), how the system makes the transition from the left-hand configuration to the right-hand configuration (Section III-C), and how the system makes the transition from the right-hand configuration back to the left-hand configuration (Section III-D).

Typically, TCP forwarding starts in the unoptimized configuration, makes a transition to the optimized configuration when the proxy shifts from control to forwarding mode, and sometimes reverts back to the unoptimized configuration should TCP forwarding go back to control mode. Note that while the connection splicing optimization is in effect, the two independent TCP connections shown on the left no longer exist on the forwarder.

B. Forwarding

The primary task of the FWD processing step shown in Fig. 2 is to change the header of incoming TCP segments to account for the differences in the two original TCP connections. Since the two TCP connections were established independently, their respective port numbers and sequence numbers are probably different. The IP addresses associated with the connections might also differ, resulting in changes that affect the IP pseudo header as well.

Fig. 3 depicts the TCP segment header; the boldface fields are those that FWD modifies. The following outlines the transformations FWD applies to each segment it forward from one connection (A) to another connection (B). For now, we ignore the problem of moving a TCP forwarder into the optimized state, and focus instead on the work involved in forwarding segments once FWD is in place. Also, we assume that the two TCP connections were established independently. If their establishment was in fact interleaved—so that one connection knew what port and sequence numbers were being used by the other connection—then additional optimizations are possible, as described in Section III-F.

- **Port Numbers:** If the TCP forwarder operates as a classical proxy, the port numbers of both TCP connections will probably differ. Therefore, the source and destination port numbers of segments arriving on A have to be changed to the port numbers of connection B. If the TCP forwarder

SrcPort			DstPort		
SeqNum					
Ack					
Hlen	Resv	Flags		AdvWin	
Checksum				UrgPtr	
Options				Padding	
Data					

Fig. 3. TCP segment header with fields modified by FWD in bold.

is a transparent proxy, this change is unnecessary because the proxy uses the same port numbers as the initiator.

- **Sequence Number:** The sequence number used by segments received by FWD on *A* are probably different from those used for segments sent by FWD on *B*. This is because TCP initializes sequence numbers randomly for each independent connection. The sequence number for an outgoing segment is computed by adding a fixed offset to the sequence number in the incoming segment.
- **Acknowledgment Number:** The acknowledgment number acknowledges the sequence numbers forwarded in the other direction. Thus the acknowledgment number in an outgoing segment is computed by subtracting from the sequence number in the incoming segment the sequence number offset for segments flowing in the other direction.
- **Checksum:** Modifying the other fields requires adjusting the TCP checksum. A constant checksum patch representing the “delta” in the checksum is used to do this efficiently. If the FWD acts as a classical proxy, the changes to the IP address fields in the IP pseudo-header are also reflected in this checksum patch.

The following pseudo code describes the changes to a segment transferred from *A* to *B*. All header fields marked **Input** represent the segment header values in the received segment. The header fields marked **Output** represent the segment header values used in the outgoing segment. Bold variables indicate constants that are part of FWD’s state. Subscripts indicate the direction for which these constants are used; e.g., $\text{SeqNumOffset}_{A \rightarrow B}$ represents the sequence number offset used to patch sequence numbers on segments received from *A* and sent to *B*.

```

Output.DstPort = RemotePortB
Output.SrcPort = LocalPortB
Output.SeqNum = Input.SeqNum + SeqNumOffsetA→B
Output.Ack = Input.Ack - SeqNumOffsetB→A
Output.Checksum = Input.Checksum + ChecksumPatchA→B.

```

The checksum calculation shown in the pseudo code is more complicated than simple addition. To account for overflows or underflows during sequence number and acknowledgment

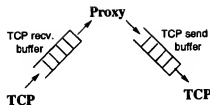


Fig. 4. TCP buffers potentially containing acknowledged data.

number calculations it is necessary to add or subtract one from the checksum. This is because the checksum is the one’s complement of the one’s complement sum of the segment.

Splicing two TCP connections significantly changes the behavior of the forwarding proxy. In the unspliced case segments sent to the proxy are acknowledged when they are processed by the incoming TCP stack. The proxy then takes responsibility for the data, resending them as necessary to ensure they reach their destination. Data are buffered in the outgoing TCP stack until they are acknowledged by the destination. When the two connections are spliced the segments no longer traverse the two TCP protocol stacks. The proxy doesn’t acknowledge data coming from the sender, nor does it resend data to the destination. Data and acknowledgments are forwarded without processing, requiring the two endpoints to handle retransmission and reordering.

Forwarding segments requires internal state in FWD. Some of this state is required to modify the header fields, such as the port numbers, sequence offsets, and checksum patch. FWD must also detect reset or termination of the TCP connection. To do so it parses the flags in the header and keeps a simplified TCP state machine. FWD also keeps one timer that is used to time-out the connections; all other TCP timers are not used.

If it is possible that the optimized forwarder will revert back to a standard forwarder, FWD also needs to store the current advertised window, the highest sequence number sent, and the highest ACK seen that will fit in the advertised window. The process of converting a spliced connection back to an unoptimized TCP forwarder is discussed in Section III-D.

C. Splicing

The header modifications required to forward a segment are relatively straight-forward. The real problem is transitioning from the unspliced state to the spliced state. The difficulty is caused by acknowledged data buffered in the forwarder. This data might be buffered by the receiving TCP’s receive buffer, within the proxy itself, and in the sending TCP’s send buffer (Fig. 4). The acknowledged data must be reliably forwarded to its destination. These data also influence the offsets calculations required by the spliced connection.

First, all data acknowledged by either connection on the unoptimized TCP forwarder must be reliably delivered to their destination. The important point is that these data have already been acknowledged by the forwarder, so it cannot depend on the source host to take responsibility for possibly retransmitting the data in the future. Thus, the forwarder must continue to run TCP—the only way it can reliably deliver data—until the currently buffered (acknowledged) data are reliably delivered to

the destination. During the time the data are being drained, however, new segments may arrive. The forwarder obviously cannot let TCP acknowledge the new data, because doing so will just give it even more data to deliver reliably, and it is impractical to wait until the two connections go idle before completing the splice. Fortunately, there are two ways to handle newly arriving segments during this transition period.

The first option is to delay the activation of the spliced connection until after the buffers have drained. During this time, the limited number of new segments that arrive are delivered to TCP so that any acknowledgment they carry can be processed, and then they are held in a separate buffer for FWD. These incoming segments are not themselves acknowledged and they are not placed in the incoming connection's receive buffer. If the buffer overflows while TCP is still processing acknowledgments, the segments are dropped after the acknowledgments have been processed. When the transition is complete, these buffered segments are processed by FWD as though they had just arrived. Again the TCP protocols are suspended as soon as all buffers are drained. This solution may drop data if the FWD buffers overflow while the TCP buffers are being drained. If the amount of data buffered in TCP is small, then the FWD buffers are unlikely to overflow.

The second option allows FWD to begin forwarding data concurrently with draining the buffers. Should any new data arrive during the transition, it is important that the original TCP protocols do not acknowledge the new data; they are only allowed to process the acknowledgments contained in those segments so that the buffers drain. In other words, all newly arriving segments are delivered to both the original TCP protocol (for acknowledgment processing only) and to FWD (for forwarding to the receiver). This solution does not drop data, but may cause data to be delivered out-of-order. This is because segments processed by FWD may be delivered before segments traversing the original TCP connections. This will not affect correctness because the destination will reorder the segments.

During the time that FWD operates concurrently with the draining process, both forwarded segments and drained segments will arrive at the destination. This means it is possible that the TCP draining buffers on the forwarder might receive an acknowledgment for a sequence number that is larger than the maximum sequence number in its send buffer. This acknowledgment is meant for both the source host and the TCP forwarder. It is most likely due to dropped ACK's or delayed ACK processing on the receiver. Since the forwarder is still processing acknowledgment in an attempt to drain its buffers, it will receive this acknowledgment too. To allow for this possibility, TCP running on the forwarder during the transition must be able to accept acknowledgments up to one full window size larger than the maximum sequence number in its send buffer.

Before the packet processing can be altered, the internal state of FWD has to be initialized, corresponding to the first step above. This requires computing the sequence number offsets ($\text{SeqNumOffset}_{A \rightarrow B}$ and $\text{SeqNumOffset}_{B \rightarrow A}$) and the checksum patches ($\text{ChecksumPatch}_{A \rightarrow B}$ and $\text{ChecksumPatch}_{B \rightarrow A}$) used by FWD. The sequence number offsets can be calculated as soon as all acknowledged data have been drained. If acknowledged data still exist in one of the

forwarder's buffers, then it is necessary to subtract the length of the buffered data from the corresponding sequence number offset. This is because the sender of a segment that is directly forwarded assumes that the buffered data were delivered, and therefore, the sequence number of the source's TCP protocol has already been increased. It is important to realize that the sequence number offset cannot be calculated earlier since we do not assume that the proxy will forward all data or add no additional data to the data stream while it is in control mode. The checksum patch can be calculated as soon as the other offsets are known since the changes in port number and IP address are already known.

D. Unsplicing

When the forwarding proxy switches from forwarding mode to control mode the connections must be unspliced. There are two complications. The first is to be able to detect that it is necessary to switch back to the unoptimized state; i.e., that the forwarder has moved from forwarding mode to control mode. The second is to correctly make the transition. The solutions to these two complications are intertwined.

It may be difficult to decide when the proxy should switch back to control mode. If the control information is sent over the spliced connection, the proxy has to monitor the data being forwarded to detect the control information. This is difficult because the FWD protocol does not reorder the segments it receives, nor does it buffer segments. The proxy has to find the control information by looking at out-of-order segments, one at a time. This makes it unlikely that the proxy will be able to filter the data to find control information. However, it seems useful to trigger a switch back to unoptimized mode as soon as data are transmitted in a certain direction. An HTTP 1.1 proxy, for example, might allow the forwarding of HTTP replies, but want to examine all (possibly pipelined) HTTP 1.1 requests. The cost of detecting this switch could vary greatly, ranging from simple monitoring if data flows in a certain direction for HTTP 1.1—which can be done by comparing a single sequence number—to maintaining a shadow state machine of the higher level protocol.

Dealing with acknowledgments makes it difficult to unsplice a connection. When the forwarder reverts to two TCP connections, it must take over handling acknowledgments. If there are no unacknowledged segments outstanding on the spliced connection, the transition back to unspliced is easy. The reconstructed TCP connections are initialized with the sequence numbers, acknowledgment numbers, and advertised window sizes stored as FWD state. The state-machine is progressed to the current state, the timers and the send window are initialized with their initial values, and a slow start is initiated. The slow start is necessary since no bandwidth estimates are available, and therefore, the congestion window sizes have to be rediscovered. In the case where no unacknowledged segments are outstanding, it is possible to stop forwarding new segments instantly.

If there are outstanding unacknowledged segments, however, the forwarder must either wait for all of them to be acknowledged—dropping data if necessary—and then switch as described above, or else it must continuously monitor the

segment stream until it has copies of all unacknowledged segments. It then uses this information to initialize the TCP connections and buffers. This solution does not drop any segments, but up to two full window sizes might have to be buffered before the switch over can be completed.

E. Flow Control

During unoptimized operation flow control is handled by the two independent TCP protocols on the forwarder, and the TCP protocol on the end hosts. During optimized operation, flow control is handled by the end hosts only; the forwarder merely drops segments, just as a congested router drops IP datagrams.

There is a complication during the transition to a spliced connection, however. Shortly after the switch to the spliced connection, the advertised window might be either too big or too small. For example, the window advertised by the destination host to the forwarder might be smaller than the window advertised by the forwarder to the source host. In this case, the source host will suddenly see a smaller advertised window after the connection is spliced, possibly triggering unnecessary retransmissions. Similarly, the send window of a host might also be bigger than the advertised window of its new peer. If so, it is likely that data will be transmitted unnecessarily. Note that RFC1122 strongly recommends¹ that the advertised window not be reduced to eliminate this unnecessary data transmission. To minimize this problem, the TCP forwarder can restrict the size of the window it advertises to the source host to the window size advertised by the destination host, minus the size of the buffered data.

More subtly, the send window of both end hosts might not represent the bandwidth of the link. If the send window is too big, the host will send too many segments and generate unnecessary congestion. However, this can only happen if traffic is extremely bursty. Otherwise, the limited buffer space available on the TCP forwarder should synchronize the send window sizes of both TCP connections.

Another issue is the increase in end-to-end round trip time (RTT) after a pair of connection have been spliced. In general, TCP's throughput decreases if the RTT increases. This is due to either an increased $RTT \times \text{bandwidth}$ product that is greater than the advertised window, or a slower increase in the sender's congestion window during the slow start or congestion avoidance phases. In either case, there are no effects on the RTT that would not have been present had the connection run end-to-end in the first place.

F. Additional Optimizations

The connection splicing optimization can be applied not only at the TCP level, but also to unfragmented IP datagrams. In addition, the optimization can be applied to the first IP fragment of an IP datagram if we allow the unfiltered forwarding of all remaining fragments, and if the MTU is large enough so that the first fragment will contain the TCP segment header.

In these two cases, the forwarder can process the IP datagrams similarly to an IP router, with the additional TCP segment header manipulation described in the previous section. Fig. 5 illustrates this scenario, which we denote as a combined IP/FWD

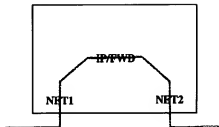


Fig. 5. FWD merged with IP. Further optimizing the spliced connection when there is no fragmentation.

processing step. The important consequence of being able to forward TCP segments at the IP level is that it makes it possible to apply any of the optimizations one might apply to an IP router. For example, if the forwarder is connected to two Ethernet, we can modify and forward the Ethernet packets directly.

Finally, under certain circumstances it is possible that the TCP forwarder can tolerate the unfiltered forwarding of all IP fragments, that is, FWD implements the identity transformation. This would happen if the unoptimized forwarder is configured as a gateway intercepting TCP connections and was careful in selecting port numbers and the starting sequence numbers when the original pair of TCP connections were opened. This being the case, FWD can be omitted and the TCP forwarder operates just like an IP router.

G. Other Issues

Additional IP-level filtering can also be done on a spliced connection. For example, to avoid attacks against the TCP stack the forwarder should limit the sequence and acknowledgment numbers of the current spliced connection to meaningful values, and drop all segments that have the SYN flag set. This is possible since all connections are established first with the proxy, not with the final destination.

As many routers already do, the forwarder can also perform network address translation (NAT). It is possible to perform NAT on spliced and unspliced connections. If, however, IP addresses are passed within the data stream, as in FTP for example, the connection has to either be spliced after the IP addresses have been altered by the proxy, or additional IP-level filters have to be added.

A final issue is TCP options. Our prototype currently supports only the MSS option, which is negotiated with the forwarder. If the MSS of both segments do not match, the ICMP Destination Unreachable message will be used to adjust the MSS after the connection is spliced. The other TCP options can be handled in much the same way as the prototype patches sequence numbers; some (e.g., SACK) don't require additional state, but others (e.g., TIMESTAMP) do.

IV. CONNECTION SPLICING IN SCOUT

Connection splicing can be implemented in any operating system; Section VI discusses an implementation in Unix. This section describes an implementation in an OS designed specifically to support communication: Scout [12]. While the primary purpose of this section is to flesh out some of the details any

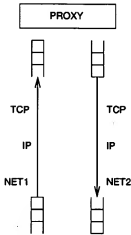


Fig. 6. TCP forwarding implemented in two Scout paths.

implementation would have to address, it has a secondary objective of illustrating how naturally a technique like connection splicing can be realized in an operating system designed around communication-oriented abstractions.

Scout is a configurable OS explicitly designed to support data flows, such as video streams through an MPEG player, or a pair of TCP connections through a firewall. Specifically, Scout defines a *path* abstraction that encapsulates data as they move through the system, for example, from input device to output device. In effect, a Scout path is an extension of a network connection through the OS. Each path is an object that encapsulates two important elements: 1) it defines the sequence of code modules that are applied to the data as they move through the system, and 2) it represents the entity that is scheduled for execution.

The path abstraction lends itself to a natural implementation of TCP forwarding. Fig. 6 schematically depicts a naive implementation of TCP forwarding (the unoptimized case) in Scout. It consists of two paths: one connecting the first network interface to the proxy and another connecting the proxy to a second network interface. In this figure, the path has a source and a sink queue, and is labeled with the sequence of software modules that define how the path “transforms” the data it carries.² To a first approximation, the configuration of Scout shown in Fig. 6 represents the implementation one would expect in a traditional OS.

The two-path configuration shown in Fig. 6 has suboptimal performance because it requires a handoff of each incoming segment from the first path to the proxy, and then from the proxy to the second path. In Scout, the entire device-to-device data flow can be encapsulated in a single path (Fig. 7). This is the implementation of choice for the unoptimized TCP forwarding case in Scout.

Connection splicing is then implemented within the same framework. Fig. 8 illustrates the two optimized configurations discussed in Section III: the path on the left corresponds to the right-hand case from Fig. 2, while the path on the right corresponds to the case shown in Fig. 5. Note that the right-hand path looks very much like an IP router would in Scout.

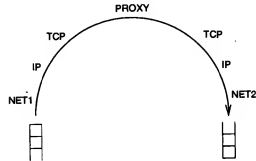


Fig. 7. TCP forwarding implemented in a single Scout path.

Looking at the implementation in a bit more detail, each path consists of a linked list of *stages*, where each module that the path traverses contributes a stage to the path during path creation. Abstractly, each stage contains the path-specific code and state for the corresponding module; e.g., the TCP control block is contained in the TCP stage of the paths shown in Figs. 6 and 7. When the proxy in an unoptimized TCP forwarding path detects a transition to forwarding mode, it does five things:

- Stops processing incoming segments and allows segments to accumulate in the path's input queue.
- Unlinks the two TCP stages and the proxy stage from the path and replaces them with a preliminary FWD stage.
- Continues processing incoming segments and data in the TCP buffers until the TCP buffers are drained.
- Unlinks the preliminary FWD stage and replaces it with the final FWD stage.
- Continues processing incoming segments.

The difference between the preliminary FWD stage and the final FWD stage is that the former forwards the segments and reliably drains the TCP buffers, whereas the latter only adjusts segment header fields.

One subtlety is that there are seldom any segments queued *within* the path that need to be drained: Scout is nonpreemptable, so in practice once a segment is removed from the input queue it is processed completely and deposited in the output queue. The only time a segment gets buffered in the middle of a path is when the scheduler selects the path for execution, the segment makes it as far as the outgoing TCP stage, but the advertised window on the second connection is closed. It would be possible to take the outgoing window into account when making the scheduling decision—i.e., not schedule a TCP forwarding path until it was certain that the segment could make it all the way to the output queue—but the consequence is that the segment would remain in the input queue, and thus, not acknowledged on the incoming TCP connection.

There is one final issue to consider: how Scout classifies each incoming packet to determine the path to which it belongs. Scout classifies packets by inspecting various header fields, such as ETH's *type* field, IP's *protnum* field, and TCP's *port* fields. While the details are beyond the scope of this paper, the relevance to connection splicing is that even after an unoptimized TCP forwarding path is spliced, the classification machinery remains the same. In other words, the spliced path no longer does any TCP processing, but the TCP port fields are still used to classify packets for the spliced path.

²As in Section III, we focus on data flowing in one direction. In reality, Scout paths, like TCP, support bidirectional data flows.

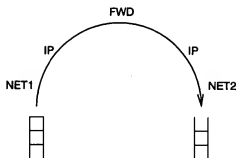


Fig. 8. Connection spliced paths in Scout.

V. PERFORMANCE

This section provides measurements of the effect of connection splicing on TCP forwarding. To make the study concrete—and to give us an existing system against which we can compare our approach—we focus on a simple firewall configuration. The proxy in the firewall does not perform any processing in control mode; it is always in forwarding mode.

A. Test Cases

We measured the following configurations of Scout:

- **2-Path:** This is a full blown TCP forwarder, as depicted in Fig. 6. This TCP forwarder uses two separate TCP paths, containing two entirely independent TCP state machines, that meet at the proxy—one to each network device. As going from one path to another often will require a context switch, this configuration is the closest to the structure of a firewall in a regular operating system like Unix or NT.
- **1-Path:** This is the configuration shown in Fig. 7. This case is similar to the 2-path configuration, except the two network devices are connected by a single path. This is the natural way of expressing a TCP forwarder in Scout. Note that this configuration still involves two TCP connections implemented by two independent TCP state machines but a single Scout path.
- **FWD:** This is an optimized version of 1-Path. Here the TCP connections have been spliced into a single connection, and the forwarder is reduced to updating the TCP headers. This configuration still supports reassembly of IP packets. This case corresponds to the left-hand configuration in Fig. 8.
- **IP/FWD:** This is a further optimized version of FWD. The network level packets are modified directly and forwarded. As a consequence, this configuration does not support reassembly of IP packets. This is the case corresponds to the right-hand configuration in Fig. 8.
- **IP Router:** This is an IP router. It also modifies network packets directly in the same way as IP/FWD, but it does not update TCP headers. It is included to show the lowest possible overhead for an intermediate host in Scout.

To compare the Scout performance with a more general-purpose operating system—so as to demonstrate that the Scout numbers are in-line with a more conventional system—we also measured the performance of a firewall and IP routing on Linux. We compiled the Linux kernel to optimize for IP routing. We consider three configurations:

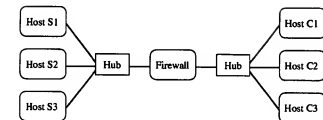
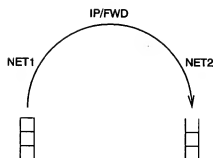


Fig. 9. Test setup.

- **TIS Firewall:** The TIS firewall toolkit offers full filter functionality [13]. We have configured it to use a null filter (plug-gw).
- **Filtering IP Router:** The in-kernel Linux IP forwarding has support for filtering on IP addresses, protocol numbers and port numbers. This is the closest thing in Linux to the IP/FWD case in Scout, except Linux neither permits starting with a proxy and later dynamically switching to the spliced connection, nor updating TCP headers.
- **IP Router:** This is the basic in-kernel Linux IP forwarding with no filtering. This shows the lowest possible overhead of the Linux configuration.

Note that while there is a myth that Linux networking performance is not very good, we have not found this to be the case with recent releases. For example, the Linux IP forwarding numbers given below are better than comparable numbers reported on BSD Unix [11]. In any case, we use the Linux numbers to calibrate the baseline case; the important results are in the incremental costs of each mechanism layered on top of this baseline.

Finally, we measure the performance of two machines connected back-to-back to evaluate the overhead of injecting a third host on the network path.

All hosts used in our experiment are 200 MHz PentiumPro workstations with 256 kB cache, 128 MB ram, and Digital Fast EtherWORKS PCI 10/100 (DE500) 32-bit PCI 10/100 Mb/s adapters. The Linux version used was 2.0.30. The physical configuration of our test setup is shown in Fig. 9. To saturate the network during throughput tests, we connected three hosts on each side of the firewall. All tests are performed between a server (hosts S1 to S3) and a client (hosts C1 to C3). In the back-to-back case, the setup was modified by connecting the two hubs to each other. All servers and clients were running Scout, as the lower complexity of Scout resulted in less variation in the measurements than Linux.

TABLE I
NON-PROCESSING RELATED OVERHEAD REMOVED FROM LATENCY MEASUREMENTS

Latency		1-byte TCP Segment	1460-byte TCP segment
Back-to-Back		77.9 μ secs	243.2 μ secs
Network Interfaces	Transmission	5.2 μ secs	121.1 μ secs
	Other	9.8 μ secs	11.7 μ secs
Total		92.9 μ secs	376.0 μ secs

TABLE II
FIREWALL AND ROUTER PROCESSING PER TCP SEGMENT

Configuration		1-byte TCP segments		1460-byte TCP segments	
		Processing time (μ secs)	Speedup	Processing time (μ secs)	Speedup
Scout	2-path	68.5	—	101.1	—
	1-path	66.1	1.04	98.6	1.03
	FWD	39.0	1.76	39.5	2.56
	IP/FWD	24.0	2.85	24.0	4.21
	IP router	22.4	3.06	22.4	4.51
	TIS Firewall	83.9	—	113.0	—
Linux	Filtering IP router	27.5	3.05	29.0	3.90
	IP router	25.5	3.29	25.4	4.45

B. Results

For all configurations, we measure the per-packet processing time for small (1-byte) and large (1460-byte) segments, and the aggregate throughput achieved with multiple connections. For Scout, we also measure the time it takes to switch from unoptimized to optimized.

1) *Processing Overhead*: To measure the per-packet processing overhead, we measured the packet round-trip times for 10 000 packets, and subtracted the back-to-back latency and network interface latency. The subtracted components are summarized in Table I. The network interface latency was obtained by measuring the processing time of a packet in the IP router configuration—that is, the time from when the packet is removed from the network interface by the interrupt handler to the time it inserted into the transmit queue of the other network interface—and subtracting this time from the total latency added by the router.

Table II summarizes the processing of a single packet in the firewalls and routers for both Scout and Linux. The 1-byte numbers reveal that connection splicing achieves a considerable speedup. Most notably, the IP/FWD case is almost a factor of three faster than application-level forwarding. In terms of packets-per-second that can be processed by the firewall, this is an increase from 14 600 to 41 600. For large packets, the speedup is even greater—a factor of four. Eliminating the extra message copy and the checksum calculations required when transferring the message from one TCP connection to another accounts for the speedup.

Also note that in both the small and large message cases, the performance of the spliced connection is very close to the performance of the IP router configuration; the TCP header transformations amount to an extra 1.6 μ s of processing. This suggests that any improvement made to IP router performance will be propagated to TCP forwarding. For example, we have found that the use of polling instead of interrupts, and the addition of a

highly optimized classification algorithm, improves IP routing performance (and hence TCP forwarding) by a factor of four [3]. On a similar note, it would be interesting to wed connection splicing with hardware supported tag switching.

Comparing the Scout and the Linux numbers, we see that the 2-path case in Scout is slightly faster than the TIS firewall on Linux. IP router performance is approximately the same for the two systems. This indicates that other types of operating systems would also benefit from connection splicing. In a Linux implementation, the IP/FWD should perform close to that of the filtering IP forwarding—the updating of the TCP and IP headers would make it slightly slower. Keep in mind, however, that simple IP filtering does not permit a proxy that can sometimes operate in control mode.

2) *Aggregate Throughput*: The sustained throughput of a TCP forwarder is also a measure of its performance. The expectation is that the improved processing overhead of the optimized forwarders should allow them to support more concurrent TCP connections.

We measured the aggregate throughput of one, two, and three concurrent TCP connections over each configuration. Each TCP connection is between a client and a server from our test setup, such that each host supports only one TCP connection. The data unit transmitted by the client process was 1460 bytes. The aggregate throughput was obtained by adding the average throughput over the last 10 s of the individual connections. This was done when the throughput had reached a stable state. Not surprisingly, these measurements turned out to be bounded by the bandwidth of the 100 Mbit Ethernet, i.e., regardless of the number of TCP connections the aggregate throughput was close to 10 Mbyte/s.

The more interesting question is how TCP forwarding behaves in the limit, that is, what bandwidth it can sustain. We can derive these numbers from the per-packet processing times presented in the previous section. For the 2-path and the IP/FWD configurations, we calculated the maximum throughput for different TCP acknowledgment patterns—either an acknowledgment

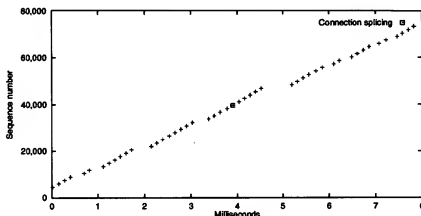


Fig. 10. TCP sequence number trace showing the effects of the Scout implementation of splicing.

TABLE III
ESTIMATED MAXIMUM THROUGHPUT OF FIREWALL IN Mbyte/s

Configuration	Message to acknowledgement ratio		
	3:1	2:1	1:1
Scout 2-path	11.7	10.8	8.6
IP/FWD	45.6	40.6	30.4

ment is sent for every third, second, or single segment. For example, if an acknowledgment is sent for every third segment, the processing requirements for the data in the three segments would be three times the processing of a 1460-byte segment, plus the processing of a single empty segment. In our case, we have approximated the empty segment with a 1-byte segment.

The results are shown in Table III. The 2-path TCP forwarder in our measurements is operating at almost maximum bandwidth of a 100 Mbit Ethernet, whereas the IP/FWD configuration is capable of supporting up to four times the bandwidth, corresponding to two full duplex OC-3 ATM connections.

3) *Cost of Splicing*: The next question is how long it takes to splice a forwarding path. Our analysis has two parts. First, we establish the base processing overhead of splicing two TCP connections together. Second, we examine the end-to-end behavior of a TCP connection sending at maximum speed when the splicing is done.

To get the basic cost, we measured the time taken to splice two idle TCP connections on a local Ethernet.³ In this case, the measurements are free of any processing that might occur due to the draining of TCP buffers. In the test we continuously opened a TCP connection, waited 15 s and closed it again. The null proxy in the firewall optimizes the path 10 s after it is established. The numbers are the average time over 1000 such optimizations. Optimizing from TCP forwarding to FWD takes 25 μ s on average. Adding the IP/FWD forwarding takes 94 μ s on average. The higher cost of switching to IP/FWD is due to the fact that Scout requires a new path creation, whereas the FWD optimization is applied to the same path by doing code substitution.

As we concurrently forward new TCP segments and empty the buffers of the old segments, the cost of performing the optimization should be small even during high load. The more

important question is whether or not the switch affects TCP's flow or congestion control algorithms. To see the effects of the switchover on a busy TCP connection, we performed the optimization 15 s into a throughput test. By tracing the sequence numbers of segments received at the server, we were unable to see any negative effects (Fig. 10).

The expected result obviously changes depending on the RTT and bandwidth of the networks involved, especially if a lot of data have to be drained over a slow link the performance will degrade due to time-outs and out of order delivery. For example, multiple out of order delivered segments might trigger the congestion window to be halved. Since the amount of data buffered by the proxy can easily be controlled by the advertised window, it is important to only buffer enough data to deal with bursts. This will minimize the impact of the splicing operation in all scenarios.

As moving to FWD forwarding reduces the processing by an average of 27.1 μ s for small messages and 59.1 μ s for large messages, it is always a good idea to switch to the FWD optimization, independent of how much data will flow over the spliced connection. Moving from FWD to IP/FWD reduces the processing by an extra 15 μ s per packet, and thus, it will take six subsequent packets to make this optimization worthwhile.

4) *Cost of Unsplicing*: The last question is how much it costs to unsplice a connection. Unsplicing has four costs associated with it. The first cost is that, in addition to fixing up the TCP header during spliced operation, FWD also has to keep track of the sequence numbers, acknowledgment numbers, and advertised windows of the spliced TCP connection. This requires some additional state and copy operations during spliced operation.

The second cost is that FWD has to determine when to unsplice. The cost of this operation depends on the application in question. It can range from simply monitoring if data flow in a certain direction by comparing two TCP sequence numbers, to mirroring an application-level state machine on the forwarder.

The third cost is the cycles required to initiate the two independent TCP state machines. The overhead of this operation is less than generating the two TCP state machines during TCP connection establishment in the first place, since the demultiplexing part of the TCP state machine is still active.

³Using a WAN instead of a local Ethernet does not alter the results of this experiment, since all operations are local and no data have to be drained. Appellant's Reply Brief EFS filed 05/30/2008

The last cost is the impact on end-to-end throughput. Since our implementation triggers slow start, the impact on the throughput would be quite significant for a high RTT and high bandwidth environment. This cost will likely dominate the cycle overhead of unsplicing a connection.

C. Buffer Requirements

Buffer size is an issue for large-scale TCP forwarders. First, just having enough memory to accommodate thousands of TCP connections can be a problem, as each connection can easily require up to 256 kB of buffering—two send buffers and two receive buffers of 64 kB each. This translates to buffer requirements of 256 Mbyte per 1000 TCP connections. As the use of persistent TCP connections is becoming more widespread, thousands of connections per TCP forwarder would not be uncommon. Splicing TCP connections together reduces the memory requirements of a TCP forwarder, since the forwarder is operating like an IP router and does not buffer segments. The only memory required for a spliced connection in addition to the memory required for a standard IP router is the FWD state used to fix up the IP addresses, port numbers, sequence numbers and checksum. This state can be stored in less than 36 bytes per connection—more than three orders of magnitude less memory than required for a typical TCP connection.

Dynamic buffer allocation is another solution to this problem, but it requires processing to determine how much buffer space to provide each connection. In this scenario, the TCP connections used for large data transfers are the most important. These TCP connections are the most likely candidates for splicing, thereby removing the buffer requirements altogether. In other words, splicing can also make the administration of a TCP forwarder easier.

VI. RELATED WORK

The idea of TCP splicing was developed independently by researchers at IBM [11], and its utility shown in supporting mobility [10]. Their work was done in the context of the Unix kernel, and so involves extensions to the socket interface. A more fundamental difference, however, is that the IBM approach is more restrictive than the one described in this paper. First, it supports splicing only at connection-setup time. Second, it allows only certain interactions among the client, proxy, and server. In particular:

- before the connection is spliced, only the client and the proxy can exchange data; the server is not allowed to send or receive data before the splice is complete;
- the proxy waits for an ACK of all data it has sent the client before engaging the splice;
- once the splice is in place, the client is allowed to send data to the server. It is the arrival of these data at the server that notify the server that the splice is complete; the server is not allowed to send until this time.

This interaction is enforced by the SOCKS library package that must be linked with both the client and the proxy [9].

In contrast, our approach allows the splicing optimization to be transparently engaged at any point in the lifetime of the

two TCP connections, including after the client and server have exchanged data. This is accomplished by having the proxy simultaneously process buffered data and forward newly arriving data, as described in Section III-C. The important consequence of this difference is that our approach allows the proxy to arbitrarily filter the data passed between the client and server before it initiates the splice and removes itself from the path. This means, for example, that a proxy is able to parse a URL in an HTTP stream. The IBM approach does not support such general filtering.

More broadly, TCP forwarders are used to separate the TCP connection on a wireless link from that of a wired network [2]. This increases performance as the characteristics of the two types of networks are very different. As a mobile host moves around, it might sometimes connect directly to a wired network, in which case the TCP forwarder becomes superfluous and can be removed. This is done in the TACO system [8], where mobile hosts can—depending on what is required from their current type of network attachment—switch between having a TCP forwarder and not, without destroying their TCP connections. The system differs from the one presented in this paper in two ways: it does not support filtering, and it uses interleaved connection establishment. This allows the TCP forwarder to be removed completely from the network path in the optimized case as no translation is necessary, but it at the same time limits the applicability of the solution. The lack of filtering makes it unsuitable for more advanced proxies such as firewalls.

Another research topic related to this paper is that of efficiently classifying packets [1], [7]. Of particular note are new algorithms to do fast routing table lookups based on variable length IP address prefixes [4], [14]. It is easy to imagine such techniques being extended to support fast IP filtering. Such an advance would be complementary to connection splicing, which can also exploit improved algorithms to determine to which path a particular packet belongs. Connection splicing is more general than IP filtering, however, since the proxy permits complex control operations.

VII. CONCLUDING REMARKS

This paper describes connection splicing, which can be applied to TCP forwarders to improve their performance. A performance study shows that an optimized TCP forwarder requires between one-half and one-quarter of the processing requirements of an unoptimized forwarder. The cost of the optimization varies according to how fast the buffers at the TCP forwarder can be emptied, but in most cases the cost is recovered within one to six packets. Furthermore, the optimization reduces the memory requirements of a TCP forwarder. The optimizations have been implemented in the Scout operating system, and it should be possible to get equivalent performance improvements in other systems.

In the future we would like to investigate when and how splicing should be applied in the emerging fields of content distribution and application-level routing. Of particular interest is the impact of splicing on persistent and SSL-secured HTTP connections.

ACKNOWLEDGMENT

The authors would like to thank the other members of the Network Systems Group at both the University of Arizona and Princeton University, and in particular, G. Tong, who implemented unsplicing. They would also like to thank the anonymous reviewers who provided helpful feedback on the manuscript.

REFERENCES

- [1] M. L. Bailey, B. Gopal, M. A. Pagels, L. L. Peterson, and P. Sarkar, "PathFinder: A pattern-based packet classifier," in *Proc. 1st Symp. Operating Systems Design and Implementation*, Monterey, CA, 1994, pp. 115–123.
- [2] A. Bakre and B. R. Badrinath, "Implementation and performance evaluation of indirect TCP," *IEEE Trans. Comput.*, vol. 46, no. 3, Mar. 1997.
- [3] A. Bavier, S. Karlin, L. Peterson, and X. Qie, "Scheduling computations on programmable routers," Princeton Univ., Princeton, NJ, Tech. Rep. 615-00, Feb. 2000.
- [4] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM'97 Symp.*, Cannes, France, Sep. 1997, pp. 3–14.
- [5] C. Brooks, M. Mazer, S. Meeks, and J. Miller, "Application-specific proxy servers as HTTP stream transducers," in *Electronic Proc. 4th Int. World Wide Web Conf., "The Web Revolution"*, Boston, MA, Dec. 1995.
- [6] M. Chatel, "RFC 1919: Classical versus transparent IP proxies," Mar. 1996.
- [7] D. Engler and M. F. Kaashock, "DPF: Fast, flexible message demultiplexing using dynamic code generation," in *Proc. ACM SIGCOMM'96 Symp.*, Stanford, CA, Aug. 1996, pp. 53–59.
- [8] J. S. Hansen, T. Reich, B. Andersen, and E. Jul, "Dynamic adaptation of network connections in mobile environments," *IEEE Internet Computing*, vol. 2, no. 1, Jan./Feb. 1998.
- [9] M. Lecch *et al.*, "SOCKS protocol version 5," RFC 1928, Mar. 1996.
- [10] D. Maltz and P. Bhagwat, "MSOCKS: An architecture for transport layer mobility," in *Proc. IEEE INFOCOM*, Apr. 1998, ftp://ftp.monarch.cs.cmu.edu/pub/dmaltz/msocks-infocom98.ps.gz, pp. 1037–1045.
- [11] —, "TCP splicing for application layer proxy performance," IBM, ftp://ftp.cs.cmu.edu/user/dmaltz/Doc/splice-perf-tr.ps, Mar. 1998.
- [12] D. Mosberger and L. Peterson, "Making paths explicit in the Scout operating system," in *Proc. 2nd Symp. Operating Systems Design and Implementation*, Oct. 1996, pp. 153–168.
- [13] M. K. Ranum and F. M. Avolio, "A toolkit and methods for Internet firewalls," in *Proc. Summer 1994 USENIX Conf.*, Berkeley, CA, USA, June 1994, pp. 37–44.
- [14] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. ACM SIGCOMM'97 Symp.*, Cannes, France, Sep. 1997, pp. 25–38.

Oliver Spatscheck was born in Germany. He received the M.S. and Ph.D. degrees in computer science from the University of Arizona, Tucson, in 1996 and 1999, respectively.

He is a Senior Technical Staff Member at AT&T Labs-Research, Florham Park, NJ, where he works on differential services, denial of service prevention, and intelligent content distribution.

Jørgen S. Hansen (S'96) received the M.S. degree in computer science in 1996 from the University of Copenhagen, Denmark, where he is currently pursuing the Ph.D. degree, also in computer science.

He visited the University of Arizona, Tucson, for seven months in 1997–1998. His research interests include operating system support for high-speed networking, distributed systems, and mobile computing.

Mr. Hansen is a student member of ACM.

John H. Hartman (M'95) received the Sc.B. degree in computer science from Brown University, Providence, RI, in 1987, and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley, in 1990 and 1994 respectively.

He has been an Assistant Professor in the Department of Computer Science, University of Arizona, Tucson, since 1995. His research interests include distributed systems, operating systems, and file systems.

Dr. Hartman is a member of ACM.

Larry L. Peterson (SM'95) received the B.S. degree in computer science from Keaney State College, Keamey, NE, in 1979, and the M.S. and Ph.D. degrees in computer science from Purdue University, West Lafayette, IN, in 1982 and 1985 respectively.

He is a Professor of computer science at Princeton University, Princeton, NJ. Prior to joining Princeton University, he was the Head of the Computer Science Department, University of Arizona, Tucson. His research focuses on end-to-end issues related to computer networks. He has been involved in the design and implementation of x-kernel and Scout operating systems. He is a coauthor of the textbook *Computer Networks: A Systems Approach* (San Mateo, CA, Morgan Kaufman, 2000).

Dr. Peterson is the Editor-in-Chief of the *ACM Transactions on Computer Systems*. He has served on the editorial boards for *IEEE/ACM TRANSACTIONS ON NETWORKING* and the *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATION*, and on program committees for SOSP, SIGCOMM, OSDI, and ASPLOS. He is also a member of the Internet's End-to-End research group, and a fellow of ACM.

Kernel Mechanisms for Service Differentiation in Overloaded Web Servers

Thiemo Voigt*

Swedish Institute of Computer Science

thiemo@sics.se

Renu Tewari

IBM T.J. Watson Research Center

tewarir@us.ibm.com

Douglas Freimuth

IBM T.J. Watson Research Center

dmgfreim@us.ibm.com

Ashish Mehra

iScale Networks

ashish@iscale.net

Abstract

The increasing number of Internet users and innovative new services such as e-commerce are placing new demands on Web servers. It is becoming essential for Web servers to provide performance isolation, have fast recovery times, and provide continuous service during overload at least to preferred customers. In this paper, we present the design and implementation of three kernel-based mechanisms that protect Web servers against overload by providing admission control and service differentiation based on connection and application level information. Our basic admission control mechanism, *TCP SYN policing*, limits the acceptance rate of new requests based on the connection attributes. The second mechanism, *prioritized listen queue*, supports different service classes by reordering the listen queue based on the priorities of the incoming connections. Third, we present *HTTP header-based connection control* that uses application-level information such as URLs and cookies to set priorities and rate control policies.

We have implemented these mechanisms in AIX 5.0. Through numerous experiments we demonstrate their effectiveness in achieving the desired degree of service differentiation during overload. We also show that the kernel mechanisms are more efficient and scalable than application level controls implemented in the Web server.

*This work was partially funded by the national Swedish Real-time Systems Research Initiative (ARTES). This work was done when the author was visiting the IBM T.J. Watson Research Center.

1 Introduction

Application service providers and Web hosting services that co-host multiple customer sites on the same server cluster or large SMP are becoming increasingly common in the current Internet infrastructure. The increasing growth of e-commerce on the web means that any server down time that affects the clients being serviced will result in a corresponding loss of revenue. Additionally, the unpredictability of flash crowds can overwhelm a hosting server and bring down multiple customer sites simultaneously, affecting the performance of a large number of clients. It becomes essential, therefore, for hosting services to provide performance isolation and continuous operation under overload conditions.

Each of the co-hosted customers sites or applications may have different quality-of-service (QoS) goals based on the price of the service and the application requirements. Furthermore, each customer site may require different services during overload based on the client's identity (preferred gold client) and the application or content they access (e.g., a client with a buy order vs. a browsing request). A simple threshold based request discard policy (e.g., a TCP SYN drop mode in commercial switches/routers discards the incoming, oldest or any random connection [1]) to delay or control overload is not adequate as it does not distinguish between the individual

QoS requirements. For example, it would be desirable that requests of non-preferred customer sites be discarded first. Such QoS specifications are typically negotiated in a service level agreement (SLA) between the hosting service provider and its customers. Based on this governing SLA, the hosting service providers need to support service differentiation based on client attributes (IP address, session id, port etc.), server attributes (IP address, type), and application information (URL accessed, CGI request, cookies etc.).

In this paper, we present the design and implementation of kernel mechanisms in the network subsystem that provide admission control and service differentiation during overload based on the customer site, the client, and the application layer information.

One of the underlying principles of our design was that it should enable "early discard", i.e., if a connection is to be discarded it should be done as early as possible, before it has consumed a lot of system resources [2]. Since a web server's workload is generated by incoming network connections we place our control mechanisms in the network subsystem of the server OS at different stages of the protocol stack processing. To balance the need for early discard with that of an informed discard, where the decision is made with full knowledge of the content being accessed, we provide mechanisms that enable content-based admission control.

Our second principle was to introduce minimal changes to the core networking subsystem in commercial operating systems that typically implement a BSD-style stack. There have been prior research efforts that modify the architecture of the networking stack to enable stable overload behavior [3]. Other researchers have developed new operating system architectures to protect against overload and denial of service attacks [4]. Some "virtual server" implementations try to sandbox all resources (CPU, memory, network bandwidth) according to administrative policies and enable complete performance isolation [5]. Our aim in this design, however, was not to build a new networking architecture but to introduce simple controls in the existing architecture that could be just as effective.

The third principle was to implement mechanisms that can be deployed both on the server as well as outside the server in layer 4 or 7 switches that perform load balancing and content based routing

for a server farm or large cluster [6]. Such switches have some form of overload protection mechanisms that typically consists of dropping a new connection packet (or some random new connection packet) when a load threshold is exceeded. For content-based routing the layer 7 switch functionality consists of terminating the incoming TCP connection to determine the destination server based on the content being accessed, creating a new connection to the server in the cluster, and splicing the two connections together [7]. Such a switch has access to the application headers along with the IP and TCP headers. The mechanisms we built in the network subsystem can easily be moved to the front-end switch to provide service differentiation based on the client attributes or the content being accessed.

There have been proposals to modify the process scheduling policies in the OS to enable preferred web requests to execute as higher priority processes [8]. These mechanisms, however, can only change the relative performance of higher priority requests; they do not limit the requests accepted. Since the hardware device interrupt on a packet receive and the software interrupt for packet protocol processing can preempt any of the other user processes [3] such scheduling policies cannot prevent or delay overload. Secondly, the incoming requests already have numerous system resources consumed before any scheduling policy comes into effect. Such priority scheduling schemes can co-exist with our controls in the network subsystem.

An alternate approach is to enable the applications to provide their individual admission control mechanisms. Although this achieves application level control it requires modifications to existing legacy applications or specialized wrappers. Application controls are useful in differentiating between different clients of an application but are less useful in preventing or delaying overload across customer sites. More importantly, various server resources have already been allocated to a request before the application control comes into effect, violating the early discard policy. However, the kernel mechanisms can easily work in conjunction with application specific controls.

Since most web servers receive requests over HTTP/TCP connections, our controls are located in three different stages in the lifetime of a TCP connection.

- The first control mechanism, *TCP SYN policing*, is located at the start of protocol stack processing of the first SYN packet of a new connection and limits acceptance of a new TCP SYN packet based on compliance with a token bucket based policer.
- The next control, *prioritized listen queue*, is located at the end of a TCP 3-way handshake, i.e., when the connection is accepted and supports different priority levels among accepted connections.
- Third, *HTTP header-based connection control*, is located after the HTTP header is received (which could be after multiple data packets) and enables admission control and priority values to be based on application-layer information contained in the header e.g., URLs, cookies etc.

We have implemented these controls in the AIX 5.0 kernel as a loadable module using the framework of an existing QoS-architecture [9]. The existing QoS architecture on AIX supports policy-based outbound bandwidth management [10]. These techniques are easily portable to any OS running a BSD style network stack¹.

We present experimental results to demonstrate that these mechanisms effectively provide selective connection discard and service differentiation in an overloaded server. We also compare against application layer controls that we added in the Apache 1.3.12 server and show that the kernel controls are much more efficient and scalable.

The remainder of this paper is organized as follows: In Section 2 we give a brief overview on input packet processing. Our architecture and the kernel mechanisms are presented in Section 3. In Section 4 we present and discuss experimental results. We compare the performance of kernel based mechanisms and application level controls in Section 5. We describe related work in Section 6 and finally, the conclusions and future work in Section 7.

2 Input Packet Processing: Background

In this section we briefly describe the protocol processing steps executed when a new connection re-

¹A port to Linux is underway.

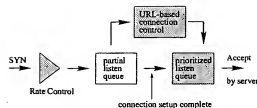


Figure 1: Proposed kernel mechanisms.

quest is processed by a web server. When the device interface receives a packet it triggers a hardware interrupt that is serviced by the corresponding device driver [11]. The device driver copies the received packet into an mbuf and de-multiplexes it to determine the queue to insert the packet. For example, an IP packet is added to the input queue, *ipintrq*. The device driver then triggers the IP software interrupt. The IP input routine dequeues the packet from the IP input queue and does the next layer de-multiplexing to invoke the transport layer input routine. For example, for a TCP packet this will result in a call to a *tcp_input* routine for further processing. The call to the transport layer input routine happens within the realm of the IP input call, i.e., there is no queuing between the IP and TCP layer. The TCP input processing verifies the packet and locates the protocol control block (PCB). If the incoming packet is a SYN request for a listen socket, a new data socket is created and placed in the partial listen queue and an ACK is sent back to the client. When the ACK for the SYN-ACK is received the TCP 3-way handshake is complete, the connection moves to an established state and the data socket is moved to the listen queue. The sleeping process, e.g., the web server, waiting on the accept call is woken up. The connection is ready to receive data.

3 Architecture Design

The network subsystem architecture adds three control mechanisms that are placed at the different stages of a TCP connection's life time. Figure 1 shows the various phases in the connection setup and the corresponding control mechanisms: (i) when a SYN packet is processed it triggers the SYN rate control and selective drop (ii) when the 3-way handshake is completed the prioritized listen queue selectively changes the ordering of accepted connections in the listen queue (iii) when the HTTP header is received the HTTP header controls decide on dropping or re-prioritizing the requests based on application

layer information. Each of these mechanisms can be activated at varying degrees of overload where the earliest and simplest control is triggered at the highest load level.

3.1 SYN Policer

TCP SYN policing controls the rate and burst at which new connections are accepted. Arriving TCP SYN packets are policed using a token bucket profile defined by the pair $\langle \text{rate}, \text{burst} \rangle$, where *rate* is the average number of new requests admitted per second and *burst* is the maximum number of concurrent new requests. Incoming connections are aggregated using specified filter rules that are based on the connection end points (source and destination addresses and ports as shown in Table 2). On arrival at the server, the SYN packet is classified using the IP/TCP header information to determine the matching rule. A compliance check is performed against the token bucket profile of the rule. If compliant, a new data socket is created and inserted in the partial listen queue otherwise the SYN packet is silently discarded.

When the SYN packet is silently dropped, the requesting client will time-out waiting for a SYN ACK and retry again with an exponentially increasing time-out value². An alternate option, which we do not consider, is to send a TCP RST to reset the connection indicating an abort from the server. This approach, however, incurs unnecessary extra overhead. Secondly, some clients send a new SYN immediately after a TCP RST is received instead of aborting the connection. Note that we drop non-compliant SYNs even *before* a socket is created for the new connection thereby investing only a small amount of overhead on requests that are dropped.

To provide service differentiation, connection requests are aggregated based on filters and each aggregate has a separate token bucket profile. Filtering based on client IP addresses is useful since a few domains account for a significant portion of a web server's requests [12]. The rate and burst values are enforced only when overload is detected and can be dynamically controlled by an adaptation agent, the details of which are beyond the scope of this paper.

²The timeout values are typically set to 6, 24, 48, up to 75 seconds.

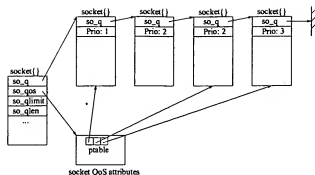


Figure 2: Implementation of the prioritized listen queue

3.2 Prioritized Listen Queue

The prioritized listen queue reorders the listen queue of a server process based on pre-defined connection priorities such that the highest priority connection is located at the head of the queue. The priorities are associated with filters (see Table 2) and connections are classified into different *priority classes*. When a TCP connection is established, it is moved from the partial listen queue to the listen queue. We insert the socket at the position corresponding to its priority in the listen queue. Since the server process always removes the head of the listen queue when calling *accept*, this approach provides better service, i.e. lower delay and higher throughput, to connections with higher priority.

Figure 2 shows the implementation of a prioritized listen queue. A special data structure used for maintaining socket QoS attributes stores an array of *priority pointers*. Each priority pointer points to the *last* socket of the corresponding priority class. This allows efficient socket insertion — a new socket is always inserted behind the one pointed to by the corresponding priority pointer.

3.3 HTTP Header-based Controls

The SYN policer and prioritized listen queue have limited knowledge about the type and nature of a connection request, since they are based on the information available in the TCP and IP headers. For web servers with the majority of the traffic being HTTP over TCP, a more informed control is possible by examining the HTTP headers. For example,

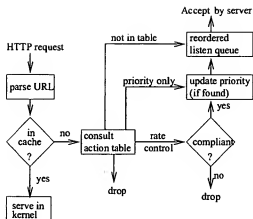


Figure 3: The HTTP header-based connection control mechanism.

Table 1: URL action table

URL	ACTION
noaccess	<drop>
/shop.html	<priority=1>
/index.html	<rate=15 conn./sec, burst=5 conn.>, <priority=1>
/cgi-bin/*	<rate=10 , burst=2>

a majority of the load is caused by a few CGI requests and most of the bytes transferred belong to a small set of large files. This suggests that targeting specific URLs, types of URLs, or cookie information for service differentiation can have a wide impact during overload.

Our third mechanism, *HTTP header-based connection control*, enables content-based connection control by examining application layer information in the HTTP header, such as the URL name or type (e.g., CGI requests) and other application-specific information available in cookies. The control is applied in the form of rate policies and priorities based on URL names and types and cookie attributes.

This mechanism involves parsing the HTTP header in the kernel and waking the sleeping web server process only after a decision to service the connection is made. If a connection is discarded, a TCP RST is sent to the client and the socket receive buffer contents are flushed.

For URL parsing, our implementation relies upon Advanced Fast Path Architecture (AFPA) [13], an

Table 2: Example Network-level Policies

(dst IP,dst port,src IP,src port)	(r,b)	priority
(*, 80, *, *)	(300,5)	3
(*, 80, 10.1.1.1, *)	(100,5)	2
(12.1.1.1, 80, *, *)	(10,1)	*

in-kernel web cache on AIX. For Linux, an in-kernel web engine called KHTTPD is available [14]. As opposed to the normal operation, where the sleeping process is woken up after a connection is established, AFPA responds to cached HTTP requests directly without waking up the server process. With AFPA, a connection is *not* moved out of the partial listen queue even after the 3-way handshake is over. The normal data flow of TCP continues with the data being stored in the socket receive buffer. When the HTTP header is received (that is when the AFPA parser finds two CR control characters in the data stream), AFPA checks for the object in its cache. On a cache miss, the socket is moved to the listen queue and the web server process is woken up to service the request.

The HTTP header-based connection control mechanism comes into play at this juncture, as illustrated in Figure 3, before the socket is moved out of the partial listen queue. The URL action table (Table 1) specifies three types of actions/controls for each URL or set of URLs. A drop action implies that a TCP RST is sent before discarding the connection from the partial listen queue and flushing the socket receive buffer. If a priority value is set it determines the location of the corresponding socket in the ordered listen queue. Finally, rate control specifies a token bucket profile of a <rate, burst> pair which drops out-of-profile connections similar to the SYN policer.

3.4 Filter Specification

A filter rule specifies the network-level and/or application-level attributes that define an aggregate and the parameters for the control mechanism that is associated with it. A network-level filter is a four-tuple consisting of local IP address, local port, remote IP address, and remote port; application-level filters were shown in Table 1. Table 2 lists some network-level filter examples. The first rule applies to the web server process listening at local port 80 on all network interfaces; it specifies that all con-

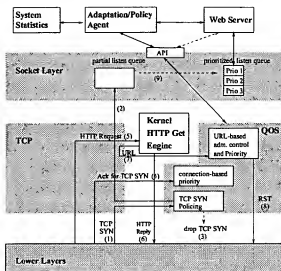


Figure 4: Enhanced protocol stack architecture.

nections to the server are rate-controlled at a rate of 300 conns/sec, a burst of 5, and a priority of 3 (the default lowest priority). The filter rules can contain range of IP addresses, wildcards, etc.

3.5 Protocol Stack Architecture

We have developed architectural enhancements for Unix-based servers to provide these mechanisms. Figure 4 shows the basic components of the enhanced protocol stack architecture, with the new capabilities utilized either by user-space agents or applications themselves. This architecture permits control over an application's inbound network traffic via policy-based traffic management [10]; an adaptation/policy agent installs policies into the kernel via a special API. The policy agent interacts with the kernel via an enhanced socket interface by sending (receiving) messages to (from) special control sockets. The policies specify filters to select the traffic to be controlled, and actions to perform on the selected traffic. The figure shows the flow of an incoming request through the various control mechanisms.

3.6 Implementation Methodology and Testbed

We have implemented the proposed kernel mechanisms in AIX 5.0, and evaluated them on the testbed

described below. As shown in Figure 4, the QoS module contains the TCP SYN policer, a priority assignment function for new connections, and the entity that performs URL-based admission control and priority assignment.

All experiments were conducted on a testbed comprising an IBM HTTP Server running on a 375 MHz RS/6000 machine with 512 MB memory, several 550 MHz Pentium III clients running Linux, and one 166 MHz Pentium Pro client running FreeBSD. The server and clients are connected via a 100 BaseT Ethernet switch. For client load generators we use Webstone 2.5 [15] and a slightly modified version of client [16]. Both programs measure client throughput in connections per second. The experimental workload consists of static and dynamic requests. The dynamic files are minor modifications of standard Webstone CGI files that simulate memory consumption of real-world CGIs.

The IBM HTTP Server is a modified Apache [17] 1.3.12 web server that utilizes an in-kernel HTTP get engine called the Advanced Fast Path Architecture (AFPA). We use AFPA in our architecture only to perform the URL parsing and have disabled any caching when measuring throughput results. Unless stated otherwise, we configured Apache to use a maximum of 150 server processes.

4 Experimental Results

4.1 Efficacy of SYN Policing

In this section we show how TCP SYN policing protects a preferred client against flash crowds or high request rates from other clients. In our setup, one client replays a large e-tailer's trace file representing a preferred customer. For the competing load we use five machines running Webstone, each with 50 clients. All clients request an 8 KB file, which is reasonable since a typical HTTP transfer is between 5 and 13 KB [12].

Without SYN policing, the e-tailer's client receives a low throughput of about 6 KB/sec. Using policing to lower the acceptance rate of Webstone clients, we expect the throughput for the e-tailer's client to increase. Figure 5 shows that the throughput for e-tailer's client increases from 100 KB/sec to 800 KB/sec as the acceptance rate for Webstone clients

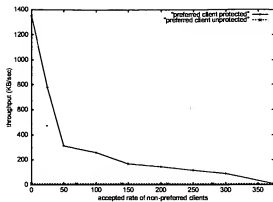


Figure 5: Throughput of the preferred e-tailer's client with and without TCP SYN policing. On the X-axis is the SYN policing rate of the non-preferred Webstone clients that are continuously generating requests. The Y-axis shows the corresponding throughput received by the e-tailer's client when there was no SYN control and when SYN control was enforced.

is lowered from 300 reqs/sec to 25 reqs/sec. The experiment demonstrates that a preferred client can be successfully protected by rate-controlling connection requests of other greedy clients.

TCP SYN policing works well when client identities and request patterns are known. In general, however, it is difficult to correctly identify a misbehaving group of clients. Moreover, as discussed below, it is hard to predict the rate control parameters that enable service differentiation for preferred clients without under-utilizing the server. For effective overload prevention the policing rate must be dynamically adapted to the resource consumption of accepted requests.

4.2 Impact of Burst Size

In the previous experiment we did not analyze the effect of the burst size on the effective throughput. The burst size is the maximum number of new connections concurrently for a given aggregate. With a large burst size, greedy clients can overload the server, whereas with a small burst, clients may be rejected unnecessarily. The burst size also controls the responsiveness of rate control. There is a tradeoff, however, between responsiveness and the achieved throughput.

We next show the effect of the burst size on the

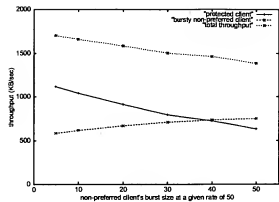


Figure 6: Impact of burst size on preferred client throughput. The burst size for policing non-preferred client is varied from 5 to 50 while the connection acceptance rate is fixed at 50 conn/sec. The plot shows the throughput achieved by the preferred and non-preferred clients along with the total throughput.

throughput of a preferred client. In our experiment, the non-preferred client is a modified client program that makes 50 to 80 back-to-back connection requests about twice a second, in addition to the specified request rate. Both the length of the incoming request burst and its timing are randomized. Figure 6 shows the throughput of preferred and non-preferred client with the SYN policing rate of the non-preferred client set to 50 conn/sec and the burst size varying from 5 to 50. The non-preferred client program requests a 16 KB dynamically generated cgi file. The preferred client is a Webstone program with 40 clients, requesting a static 8 KB file. As the burst size is increased from 5 to 50, the client's throughput increases from 36.6 conns/sec (585.6 KB/sec) to 47.7 conns/sec (752 KB/sec), while the throughput received by the preferred client decreases from about 140 conns/sec (1117 KB/sec) to 79 conns/sec.

Intuitively the overall throughput should have increased, however, the observed decrease in total throughput is due to the fact that we accept more CPU consuming CGI requests from client, thereby, incurring a higher overhead per byte transferred.

4.3 Prioritized Listen Queue: Simple Priority

With TCP SYN policing, one must limit the greedy non-preferred clients to a meaningful rate during overload. In most cases it is relatively simpler to just give the preferred clients a higher absolute pri-

ority. We demonstrate next that the prioritized listen queue provides service differentiation, especially with a large listen queue length.

In our experiments we classify clients into three priority levels. Clients belonging to a common priority level are all created by a Webstone benchmark that requests an 8 KB file. A separate Webstone instance is used for each priority level. We measure client throughput for each priority level while varying the total number of clients in each class. Each priority class uses the same number of clients.

In the first experiment, the Apache server is configured to spawn a maximum of 50 server processes. The results in Figure 7 show that when the total number of clients is small, all priority levels achieve similar throughput. With fewer clients, server processes are always free to handle incoming requests. Thus, the listen queue remains short and almost no reordering occurs. As the number of clients increases, the listen queue builds up since there are fewer Apache processes than concurrent client requests. Consequently, with re-ordering the throughput received by the high priority client increases, while that of the two lower priority clients decreases. Figure 7 shows that with more than 30 Webstone clients per class only the high-priority clients are served while the lower-priority clients receive almost no service.

Figure 8 illustrates the effect on response times observed by clients of the three priority classes. It can be seen that as the number of clients increases across all priority classes the response time for the lower priority classes increases exponentially. The response time of the high priority class, on the other hand, only increases sub-linearly. When the number of high priority requests increases, the lower priority ones are shifted back in the listen queue, thereby, increasing their response times. Also as more high priority requests get serviced by the different server processes running in parallel and competing for the CPU their response times increase.

We also observed that when the number of high priority requests was fixed and the lower priority request rate was steadily increased, the response time of the high priority requests remained unaffected.

The priority-based approach enables us to give low delay and high throughput to preferred clients independent of the requests or request patterns of

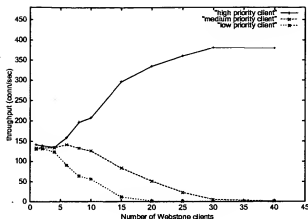


Figure 7: Throughput with the prioritized listen queue and 3 priority classes with 50 Apache processes. The number of clients in each class remains equal.

other clients. However, one may need many priority classes for different levels of service. The main drawback of a simple priority ordering is that it provides no protection against starvation of low-priority requests.

4.4 Combining Policing and Priority

To prevent starvation, low priority requests need to have some minimum number reserved slots in the listen queue so that they are not always preempted by a high priority request. However, reserving slots in the listen queue arbitrarily could cause a high priority request to find a full listen queue, which would in turn cause it to be aborted after its 3-way handshake is completed. To avoid starvation with fixed priorities, we combine the listen queue priorities with SYN policing to give preferred clients higher priority, but limiting their maximum rate and burst, thereby, implicitly reserving some slots in the queue for the lower priority requests.

Table 3 shows the results for experiments with three sets of Webstone clients with different priorities and rate control of the high priority class. The lower priority class has 30 Webstone clients while the high priority class has 150 Webstone clients spread over three different hosts. With no SYN policing of the clients in the high priority class, the two low-priority clients are completely starved. Table 3 shows that rate limiting the clients in the high priority class to 300 conn/sec prevents starvation; the medium and

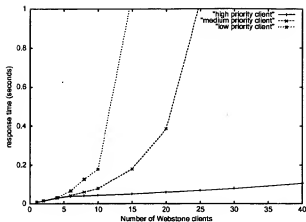


Figure 8: Response time with the prioritized listen queue and 3 priority classes with 50 Apache processes. The number of clients in each class remains equal.

Table 3: TCP SYN policing of a high-priority client to avoid starvation of other clients.

Throughput (conn/sec) of each priority class			
client priority	(rate, burst) limit of high priority		
	none	(300,300)	(200,200)
high	381	306	196
medium	0	78.6	180
low	0	4.1	13

low priority clients achieve a throughput of 78.6 and 4.1 conn/sec respectively.

4.5 HTTP Header-based Connection Control

In this section we illustrate the performance and effectiveness of admission control and service differentiation based on information in the HTTP headers i.e., URL name and type, cookie fields etc.

Rate control using URLs: In our experimental scenario the preferred client replaying the e-tailer's trace needs to be protected from overload due to a large number of high overhead CGI requests from non-preferred clients. The client issuing CGI requests is an sclient program requesting a dynamic file of length 5 KB at a very high rate. Figure 9 shows that without any protection, the preferred e-tailer's customer receives a low throughput of under 1 KB/sec. By rate-limiting the dynamic requests

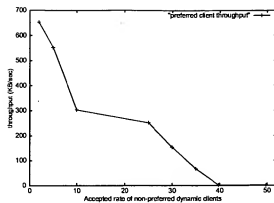


Figure 9: URL-based policing to protect preferred e-tailer's customers. The graph shows the resulting throughput of the preferred e-tailer's client as a specific high overhead CGI requests is limited to a given number of conn/sec

from 40 reqs/sec to 2 reqs/sec the throughput of the preferred e-tailer's customer improves from 1 KB/sec to 650 KB/sec. In contrast to TCP SYN policing (Figure 5), URL rate control targets a specific URL causing overload instead of a client pool.

URL priorities: In this section we present the results of priority assignments in the listen queue based on the URL name or type being requested. The clients are Webstone benchmarks requesting two different URLs, both corresponding to files of size 8 KB. There are two priority classes in the listen queue based on the two requested URLs. Figure 10 shows that the lower priority clients (accessing the low priority URL) receive lower throughput and are almost starved when the number of clients requesting the high priority URL exceeds 40. These results correspond to the results shown earlier with priorities based on the connection attributes (see Figure 7). The average total throughput, however, is slightly lower with URL-based priorities due to the additional overhead of URL parsing.

Combined URL-based rate control and priorities: To avoid starvation of requests for the low-priority URL, we rate limit the requests for the high-priority URL. In this experiment, we assign a higher priority to requests for a dynamic CGI request of size 5 KB (requested by an sclient program), and lower priority to requests for a static 8 KB file (requested by the Webstone program). Table 4 shows that starvation can be avoided by rate-limiting the high-priority URL requests.

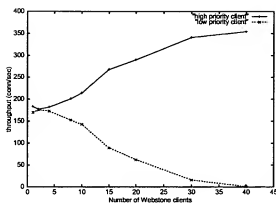


Figure 10: Throughput with 2 URL-based priorities and 50 Apache server processes. The number of clients in each class is equal

Table 4: URL-based policing of a high-priority client to avoid starvation of other clients.

client priority	Throughput (conn/sec)		
	(rate, burst) limit of high priority		
	none	(30,10)	(10,10)
high	61.7	29.0	10.1
low	0	10.2	117

4.5.1 Overload Protection from High Overhead Requests

So far we have used the URL-based controls for providing service differentiation based on URL names and types. In the next experiment, we investigate if URL-based connection control can be used to protect a web server from overload by a targeted control of high overhead requests (e.g., CGI requests that require large computation or database access).

We use the `scilent` load generator to request a given high overhead URL and control the request rate, steadily increasing it and measuring the throughput. Figure 11 shows the client's throughput with varying request rates for a dynamic CGI request that generates a file size of 29 KB. The throughput increases linearly with the request rate up to a critical point of about 63 connections/sec. For any further increase in the request rate the throughput falls exponentially and later plateaus to around 40 connections/sec. To understand this behavior we used `vmstat` to capture the paging statistics. Since the dynamic requests are memory-intensive, the available free memory rapidly declines. For some combinations of the request rate

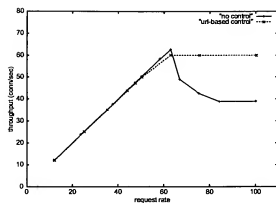


Figure 11: Overload protection from high overhead requests using URL-based connection control. The graph shows the throughput of web server with no controls servicing CPU intensive CGI requests and the corresponding throughput when the CGI requests are limited to 60 reqs/sec.

and the number of active processes, the available free memory falls to zero. Eventually the system starts thrashing as the CPU spends most of the time waiting for pending local disk I/O. In the above experiment with 150 server processes and a request rate of 63 reqs/sec the wait time starts increasing as indicated by the `wait` field of the output from `vmstat`.

To prevent overload we use URL-based connection control to limit the number of accepted dynamic CGI requests to a rate of 60 reqs/sec and a burst of 10. The dashed line in Figure 11 shows that with URL-based control the throughput stabilizes to 60 reqs/sec and the server never thrashes. In the above experiment, the URL-based connection control can handle request rates of up to 150 requests per second. However, for request rates beyond that thrashing starts as the kernel overhead of setting up connections, parsing the URL and sending the RSTs, becomes substantial.

To further delay the onset of thrashing we augment the URL-based control with the TCP SYN policer. For every TCP RST that is sent we drop any subsequent SYN request from that same client for a specified time interval. The time interval selected is the timeout value used for a lost SYN.

Table 5: Performance of AFPA and matching a URL to a rule for a 8 KB file with different URL lengths.

Throughput (conn/sec)			
URL off length	AFPA (no cache)	AFPA on, (no cache) no rule	AFPA on, matching rule
11 char.	370.1	340.5	338.3
80 char.	361.5	321.9	319.4
160 char.	355.1	321.1	303.7

Table 6: Overhead of kernel mechanisms

Operation		Cost(μ sec)
TCP SYN policing	1 filter rule	7.9
	3 filter rules	9.6
classification and priority	1 rule	4.4
	3 rules	5.0
AFPA including URL parsing		19
URL-based rate control including URL matching	1 rule	5.0
	2 rules	5.8
	3 rules	6.5
URL-based priority including URL matching	1 rule	3.8
	2 rules	4.1
	3 rules	4.3

4.5.2 Discussion

The HTTP header-based rate control relies on sending TCP RST to terminate non-preferred connections as and when necessary. In a more user-friendly implementation we could directly return an HTTP error message (e.g., server busy) back to the client and close the connection.

Our current implementation of URL-based control handles only HTTP/1.0 connections. We are currently exploring different mechanisms for HTTP/1.1 with keep-alive connections to limit the number and types of requests that can be serviced on the same persistent connection. The experiments in the previous section have only presented results on URL based controls. Similar controls can be set based on the information in cookies that can capture transaction information and client identities.

4.6 Overhead of the Kernel Mechanisms

We quantify the overhead of matching URLs in the kernel for varying URL lengths. Table 5 shows that the overhead of matching a URL to a rule is moderate (under 6% for a 160 character URL). The throughput numbers are for 20 Webstone clients requesting an 8 KB file. Rules are matched using the standard string comparison (`strcmp`) with no optimizations; better matching techniques can reduce this overhead significantly. On a cache miss, the in-kernel AFPA cache introduces an overhead of about 10% for an 8 KB file. However, the AFPA cache under normal conditions increases performance significantly for cache hits. In our experiments we have the cache size set to 0 so that AFPA cannot serve any object from the cache. When caching is enabled Webstone received a throughput of over 800 connections per second on a cache hit.

Table 6 summarizes the additional overhead of the implemented kernel mechanisms. The overhead of compliance check and filter matching for TCP SYN policing with 1 filter rule is 7.9 μ secs. Simply matching the filter, allocating space to store QoS state, and setting the priority adds an overhead of around 4.4 μ secs for 1 filter rule. The policing controls are more expensive as they include accessing the clock for the current time. Surprisingly, the URL matching and rate control has a low overhead of 5.0 μ secs for a URL of 11 chars. This happens to be lower than SYN policing as the `strcmp` matching is cheaper for one short URL compared to matching multiple IP addresses and port numbers. The overhead of URL matching and setting priorities for a single rule is around 3.8 μ secs. The most expensive operation is the call to AFPA to parse the URL. AFPA not only parses the URL, but also does logging, checks if the requested object is in the network buffer cache, and pre-computes the HTTP response header.

5 Comparison of User Space and Kernel Mechanisms

In this section we compare the effectiveness of our kernel mechanisms with overload protection and service differentiation mechanisms implemented in user space. One might argue that kernel-based mechanisms are less flexible and more difficult to implement than mechanisms implemented in user space. User level controls although limited in their capa-

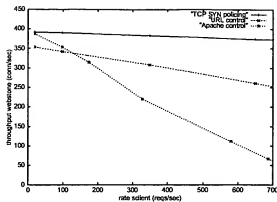


Figure 12: Throughput of kernel-based TCP SYN policing, kernel based URL rate control and Apache module based connection rate control. The throughput achieved by Webstone clients is measured against an increasing request load generated by client. The client requests are rate controlled to 10.0 req/sec with a burst of 2.

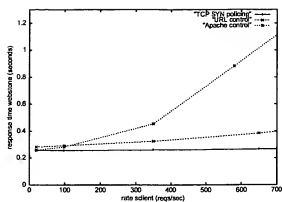


Figure 13: Response times using kernel-based TCP SYN policing, kernel based URL rate control and Apache module based connection rate control. The response time achieved by Webstone clients is measured against an increasing request load generated by client. The client requests are rate controlled to 10 req/sec with a burst of 2.

bilities, have easy access to application layer information. However, kernel mechanisms are more scalable and provide much better performance. In general, placing mechanisms in the kernel is beneficial if it leads to considerable performance gains and increases the robustness of the server without relying on the application layer to prevent overload.

To enable a fair comparison we have extended the Apache 1.3.12 server with additional modules [18] that police requests based on the client IP address and requested URL. The implemented rate control schemes use exactly the same algorithms as our kernel based mechanisms. If a request is not compliant we send a "server temporarily unavailable" (503 response code) back to the client and close the connection.

The experimental setup consists of a Webstone traffic generator with 100 clients requesting a file of size 8 KB along with an client program requesting a file of size 16 KB. The client's requests are rate controlled with a rate of 10 requests per second and a burst of 2; there are no controls set for the Webstone clients. During our experiments, we steadily increased the client's request rate.

Figure 12 illustrates that when the request load of the client program is low (20 reqs/sec), the Webstone throughput is 392 conn/sec and 387.3 conn/sec for TCP SYN policing and Apache user level controls respectively. These controls limit the client acceptance rate to 10.0 conn/sec. With in-kernel

URL-based rate control the throughput is lower (354 conn/sec). This low throughput is caused by the additional 10% overhead added by AFPA (with no caching) as discussed in Section 4.6. As discussed earlier, with the cache size set to zero, we add more overhead than necessary for URL parsing, without the corresponding gains from AFPA caching.

As the client's request load increases further, TCP SYN policing is able to achieve a sustained throughput for the Webstone clients, while the Apache based controls shows a marked decline in throughput. The graph shows that for a client load of 650 reqs/sec the Webstone throughput for TCP SYN policing is 374 conn/sec; for in-kernel URL-based connection control it is 260.7 conn/sec; for Apache user level controls the throughput sinks to about 75 conn/sec. The corresponding results for response times are shown in Figure 13.

The experiment demonstrates that the kernel mechanisms are more efficient and scalable than the user space mechanisms. There are two main reasons for the higher efficiency and scalability: First, non-compliant connection requests are discarded earlier reducing the queuing time of the compliant requests, in particular less CPU is consumed and the context switch to user space is avoided. Second, when implementing rate control at user space, the synchronization mechanisms for sharing state among all the Apache server processes decrease performance.

6 Related Work

Several research efforts have focused on admission control and service differentiation in web servers [19], [20], [21], [22], [8] and [23]. Almeida *et al.* [8] use priority-based schemes to provide differentiated levels of service to clients depending on the web pages accessed. While in their approach the application, i.e., the web server, determines request priorities, our mechanisms reside in the kernel and can be applied without context-switching to user level. *WebQoS* [23] is a middleware layer that provides service differentiation and admission control. Since it is deployed in user space, it is less efficient compared to kernel-based mechanisms. While *WebQoS* also provides URL-based classification, the authors do not present any experiments or performance considerations. Cherkasova *et al.* [20] present an enhanced web server that provides session-based admission control to ensure that longer sessions are completed. Crovella *et al.* [24] show that client response time improves when web servers serving static files serve shorter connections before handling longer connections. Our mechanisms are general and can easily realize such a policy.

Reumann *et al.* [25] have presented virtual services, a new operating system abstraction that provides resource partitioning and management. Virtual services can enhance our scheme by, for example, dynamically controlling the number of processes a web server is allowed to fork. In [26] Reumann *et al.* have described an adaptive mechanism to setup rate controls for overload protection. The receiver livelock study [2] showed that network interrupt handling could cause server livelocks and should be taken into consideration when designing process scheduling mechanisms. Banga and Druschel's [27] *resource containers* enable the operating system to account for and control the consumption of resources. To shield preferred clients from malicious or greedy clients one can assign them to different containers. In the same paper they also describe a multi listen socket approach for priorities in which a filter splits a single listen queue into multiple queues from which connections are accepted separately and accounted to different principals. Our approach is similar, however, connections are accepted from the same single listen queue but inserted in the queue based on priority. Kanodia *et al.* [21] present a simulation study of queuing-based algorithms for admission control and service differentiation at the front-end. They focus

on guaranteeing latency bounds to classes by controlling the admission rate per class. Aron *et al.* [28] describe a scalable request distribution architecture for clusters and also present resource management techniques for clusters.

Scout [29], Rialto [30] and Nemesis[31] are operating systems that track per-application resource consumption and restrict the resources granted to each application. These operating systems can thus provide isolation between applications as well as service differentiation between clients. However, there is a significant amount of work involved to port applications to these specialized operating systems. Our focus, however, was not to build a new operating system or networking architecture but to introduce simple controls in the existing architecture of commercial operating systems that could be just as effective.

7 Conclusions and Future Work

In this paper, we have presented three in-kernel mechanisms that provide service differentiation and admission control for overloaded web servers. TCP SYN policing limits the number of incoming connection requests using a token bucket policer and prevents overload by enforcing a maximum acceptance rate of non-preferred clients. The prioritized listen queue provides low delay and high throughput to clients with high priority, but can starve low priority clients. We show that starvation can be avoided by combining priorities with TCP SYN policing. Finally, URL-based connection control provides in-kernel admission control and priority based on application-level information such as URLs and cookies. This mechanism is very powerful and can, for example, prevent overload caused by dynamic requests. We compared the kernel mechanisms to similar application layer controls added in the Apache server and demonstrated that the kernel mechanisms are much more efficient and scalable than the Apache user level controls.

The kernel mechanisms that we presented rely on the existence of accurate policies that control the operating range of the server. In a production system it is unrealistic to assume knowledge of the optimal operating region of the server. We are currently implementing a policy adaptation agent (Figure 4) that dynamically adapts the rate control policies to the changing workload conditions. This adaptation

agent uses available kernel statistics and past history to select appropriate values for the various policies and monitors the interaction between various control options on the overall performance during overload.

Our current implementation does not address security issues of fake IP addresses and client identities. We plan to integrate a variety of overload prevention policies with traditional firewall rules to provide an integrated solution.

References

- [1] "Cisco TCP intercept," <http://www.cisco.com>.
- [2] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," in *Proc. of USENIX Annual Technical Conference*, Jan. 1996.
- [3] P. Druschel and G. Banga, "Lazy receiver processing (LRP): a network subsystem architecture for server systems," in *Proc. of OSDI*, Oct. 1996, pp. 91-105.
- [4] O. Spatscheck and L. Peterson, "Defending against denial of service attacks in scout," in *Proc. of OSDI*, Feb. 1999.
- [5] "Ensim corporation: virtual servers," <http://www.ensim.com>.
- [6] "Alteon web systems," <http://www.alteonwebsystems.com>.
- [7] "Cisco arrowpoint web network services," <http://www.arrowpoint.com>.
- [8] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing differentiated levels of service in web content hosting," in *Proc. of Internet Server Performance Workshop*, Mar. 1999.
- [9] T. Barzilai, D. Kandlur, A. Mehra, and D. Saha, "Design and implementation of an rsvp based quality of service architecture for an integrated services internet," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, pp. 397-413, Apr. 1998.
- [10] A. Mehra, R. Tewari, and D. Kandlur, "Design considerations for rate control of aggregated tcp connections," in *Proc. of NOSSDAV*, June 1999.
- [11] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated, Volume 2*, Addison-Wesley Publishing Company, 1995.
- [12] Martin F. Arlitt and Carey I. Williamson, "Web server workload characterization: The search for invariants," in *Proc. of ACM Sigmetrics*, Apr. 1996.
- [13] P. Joubert, R. King, R. Neves, M. Russinovich, and J. Tracey, "High performance memory based web caches: Kernel and user space performance," in preparation.
- [14] "Khttpd - linux http accelerator," <http://www.fearus.demon.nl/>.
- [15] "webstone," <http://www.mindcraft.com>.
- [16] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *Proc. of USITS*, Dec. 1997.
- [17] "apache," <http://www.apache.org>.
- [18] L. Stein and D. MacEachern, *Writing Apache modules with Perl and C*, O'Reilly, 1999.
- [19] T. Abdelzaher and N. Bhatti, "Web server qos management by adaptive content delivery," in *Intl. Workshop on Quality of Service*, June 1999.
- [20] L. Cherkasova and P. Phaal, "Session based admission control: a mechanism for improving the performance of an overloaded web server," Tech. Rep., Hewlett Packard, 1999.
- [21] V. Kanodia and E. Knightly, "Multi-class latency-bounded web servers," in *Intl. Workshop on Quality of Service*, June 2000.
- [22] K. Li and S. Jamin, "A measurement-based admission controlled web server," in *Proc. of INFO-COMM*, Mar. 2000.
- [23] Nina Bhatti and Rich Friedrich, "Web server support for tiered services," *IEEE Network*, Sept. 1999.
- [24] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection scheduling in web servers," in *Proc. of USITS*, Oct. 1999.
- [25] J. Reumann, A. Mehra, K. Shin, and D. Kandlur, "Virtual services: A new abstraction for server consolidation," in *Proc. of USENIX Annual Technical Conference*, June 2000.
- [26] H. Jamjoom and J. Reumann, "Qguard: protecting internet servers from overload," Tech. Rep., University of Michigan, CSE-TR-427-00, 2000.
- [27] G. Banga, P. Druschel, and J. Mogul, "Resource containers: a new facility for resource management in server systems," in *Proc. of OSDI*, Feb. 1999.
- [28] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable content-aware request distribution in cluster-based network servers," in *Proc. of USENIX Annual Technical Conference*, June 2000.
- [29] D. Mosberger and L. L. Peterson, "Making paths explicit in the scout operating system," in *Proc. of OSDI*, Oct. 1996, pp. 153-167.
- [30] M. B. Jones, J. S. Barrera III, A. Forin, P. J. Leach, D. Rosu, and M. Rosu, "An overview of the Rialto real-time architecture," in *ACM SIGOPS European Workshop*, Sept. 1996, pp. 249-256.
- [31] Thiemo Voigt and Bengt Ahlgren, "Scheduling TCP in the Nemesis operating system," in *IFIP WG 6.1/WG 6.4 International Workshop on Protocols for High-Speed Networks*, Aug. 1999.

TRIAD: A New Next-Generation Internet Architecture

Computer Science Department
Stanford University
{cheriton,mgritter}@dsg.stanford.edu

Abstract

Today, the primary use of the Internet is content distribution — delivery of web pages, audio and video streams to client browsers. However, scaling to meet the enormous demands of the web have required *ad hoc* and, in some cases, proprietary protocols and mechanisms to be deployed. Unfortunately, these *ad hoc* mechanisms have scaling problems and conflict with the original Internet architecture. IPv6, the current leading candidate for a next generation Internet architecture, provides more addresses but does not help with the content problem, given that its design predates the web.

In this paper, we present TRIAD as a new next generation architecture. A key aspect of TRIAD is the explicit inclusion of a *content layer* that provides scalable content routing, caching and content transformation. TRIAD also provides extensible path-based addressing using a simple “shim” protocol on top of IPv4. We claim that TRIAD not only provides scalable content distribution, but also solves the Internet problems with supporting *network address translation (NAT)* and provides innovative solutions to mobility, virtual private networks, policy-based routing and source spoofing. Its compatibility with IPv4, TCP, DNS and other dominant Internet protocols facilitates incremental deployable.

1 Introduction

With the emergence of the web, the primary use of the Internet is content distribution, i.e. web pages, and increasingly audio and video streams. Some measurements indicate that 70 to 80 percent of Internet traffic is HTTP traffic (and most of the rest of the traffic is routing and DNS). That is, almost all of the traffic in the wide area is delivery of content, and ancillary traffic to locate it and determine how to deliver it. Today, millions of clients are accessing thousands of web sites on a daily basis, with the top 20 web sites supplying about 10 percent of the content. Moreover, new popular web sites and temporarily attractive web sites can prompt the arrival of a so-called *flash crowd* of clients, often overwhelming the resources of the associated web site.

To scale content delivery to support these demands, a variety of *ad hoc* and, in some cases, proprietary mechanisms have been deployed. For instance, content is geographically replicated at multiple sites with specialized name servers that redirect DNS lookups to nearby (to the client) sites based on specialized routing, load monitoring and Internet “mapping” mechanisms, so-called *content routing*. Further, proxies in the network provide transparent caching of content, further reducing the load on web sites and the network resources between the cache and the web site. Finally, load-balancing switches at each web site allow the *virtual host* for the web site to be realized as N physical hosts, distributing the content requests across these hosts based on individual server load and the content it holds. Future extensions are expected to provide content transformations proxies that will transform content to a representation suitable for the requesting client, such as a mobile PDA.

These mechanisms violate the original Internet architecture in various ways, do not fully address scalable content distribution, and compromise the basic philosophy of the Internet as being based on open, community-based standards. In particular, content routing requires a DNS server that accesses some form of routing information, a layer violation (or duplication of the routing layer), yet still requires the world-wide clients of a site to access a single centralized server as part of accessing that site. This roundtrip to a central server is quickly becoming the dominant performance issue for clients as Internet data rates move to multiple gigabits, reducing the transfer time for content to insignificance. Transparent caching requires hijacking

transport-level connections, violating the end-to-end semantics and causing connection failures when the routing changes to route around the cache. Moreover, these caches introduce extra delay in connection setup for non-cacheable content while they collect the HTTP header information required to determine whether the content is cacheable or not. Load balancing switches rely on *network address translation (NAT)* which requires rewriting addresses, port numbers, transport-layer checksums and even packet data, making mockery of the original end-to-end semantics. (Network Address Translation [11, 12] (NAT) is also being widely deployed for other reasons, such as *address allocation autonomy*, multi-homing, concealing endpoints and increasing the number of addresses available in an edge network.

In 1992, work on IPng, next-generation IP, was initiated based on concern that the Internet was running out of addresses. This effort, predating the web, focused on providing more addresses but failed to anticipate the strong emergence of content as the primary use of the web and thus failed to address the content distribution issues described above.

In this paper, we present TRIAD¹, as a new next generation architecture. TRIAD defines an explicit *content layer* that provides scalable content routing, caching, content transformation and load balancing, integrating naming, routing and transport connection setup. In particular, all end-to-end identification in TRIAD is based on names and URLs, with IP addresses reduced to the role of transient routing tags. At this content layer, The integrated directory, routing and connection setup to provide efficient content routing to replicated content and eliminate roundtrip times to access the content in most cases. Finally, TRIAD supports path-based addressing using a simple "shim" protocol on top of IPv4 called WRAP, for content routing control and extensible addressing. This extensible addressing eliminates the need to transition the Internet to IPv6. We claim that TRIAD not only provides scalable content distribution, but also solves the Internet problems associated with *network address translation (NAT)*. TRIAD also provides attractive solutions to mobility, virtual private networking, policy-based routing and source spoofing. TRIAD can be incrementally deployed, initially without changes to end-hosts or applications beyond that already required for NAT.

The next section describes the TRIAD content layer. Section 3 describes the named-based identification of endpoints and its implications in more detail, illustrating with the associated modifications to TCP. Section 4 describes the integrated naming and routing in TRIAD. Section 5 describes the extensible path-based addressing using WRAP, its expected implementation and WRAPsec, an IPsec-like mechanism for end-to-end security. Section 7 discusses the implementation of TRIAD and some measurements to evaluate it we have made to date. Section 6 describes the TRIAD approaches to mobility, VPNs, policy-based routing and extended forwarding checks. Section 8 describes an extended NAT router that allows incremental deployment of TRIAD. Section 9 describes the incremental deployment of TRIAD. Section 10 describes related work, and we close with conclusions.

2 TRIAD Content Layer

At the content layer, a client either requests to access some content, identified by some name specification, to read or to write (or both). For example, a client may request the CNN news, which is delivered as some combination of web pages and audio and video streams. In contrast, lower levels of the architecture simply transmit or receive data packets.

The TRIAD content layer may return a *network pointer* through which content may be read or written, rather than the content itself, especially when the content is large or indeterminate in length.

For compatibility with the World-wide Web, the TRIAD content layer uses the Web Uniform Resource Locator (URL) as the format for content identification, optionally augmented with web "cookies". Further, this "network pointer" is realized as an HTTP/TCP connection through which the content is read or written.

The content layer is implemented by *content routers* that direct the requests towards content servers storing the content, *content servers* that provide the content services, and *content caches* that transparently store some content nearer to the client than the servers themselves. It may also include *content transformers* that transform the content from one form to another in response to characteristics of the client and its network connection. For instance, a content transformer may reformat content for presentation on a PDA, and transmission to this PDA over a limited bandwidth wireless link.

¹TRIAD is an acronym, standing for *Translating Relaying Internet Architecture integrating Active Directories*.

For example, in TRIAD, a client sends a lookup request with a URL for the CNN news page to its content router. The content router looks up this URL (typically just the DNS portion of the URL) and determines a nearby replica for this content, forwarding the request to a next hop accordingly. The next hop content router looks up this URL as well and may determine it has this content cached locally. In this case, it returns TCP connection information to the client, allowing the client to read this content from its cache over this (HTTP/TCP) connection.

In this way, the content layer explicitly supports the key requirements for scalable content distribution. In particular, it explicitly supports replicated content, leveraging the routing information to locate the nearest replica. It further provides hierarchical transparent caching, minimizing the implosion of demand on popular content servers and their Internet connections without requiring explicit configuration of proxies in the browsers while still avoiding the "route-around" danger with current transparent caches. Finally, it eliminates a roundtrip relative to the current Internet by combining the name lookup with connection setup.

The TRIAD content layer also supports multicast-based content distribution, by allowing a content lookup to subsume the functionality of the current multicast subscription, connecting client into the multicast session and the associated distributed tree.

For simplicity, one can view the DNS portion of a URL as mapping to the content server and the file name portion selecting the file within the content server. However, there is some flexibility in this partitioning of a URL into DNS name and file name portion. For instance, a prefix of the DNS name can determine a content volume within a specific content server. Similarly, a content router can direct an HTTP connection to a separate server based on the file portion. The primary constraint now is that the file portion is not available until the HTTP connection is established. For simplicity, we describe TRIAD assuming the conventional use of DNS name specifying the content server/volume, and file name portion specifying the content within this volume.

For deployability, TRIAD is designed to be highly compatible with existing Internet infrastructure, requiring extensions primarily at the directory system level. The TRIAD content layer subsumes the DNS directory system, providing DNS name lookup as a subset of its services. It also includes the wide-area routing system, allowing it to map content name to the closest replica based on the proximity information (accessed from the routing system). Finally, the content layer includes a mechanism for creating the so-called network pointers, more conventionally known as transport connections, thus subsuming the connection setup portion of TCP. The TRIAD content layer interfaces to the conventional transport and (inter)network layers of the Internet, allowing IP and TCP to be used, but with some modifications.

The following sections describe specific changes in more detail.

3 Named-based Identification Endpoints

In TRIAD, endpoints are identified by name. At the content level, the endpoint is a source of content and is identified by a URL. At the host interface level² the endpoint is identified by a hierarchical character-string (DNS) name. For example, "pescadero.stanford.edu" identifies a host interface on a host at Stanford. In contrast, an address for this interface may change over time, even during the lifetime of transport layer connections.

This name is the basis for all end-to-end identification, authentication, and reference passing. There is no other identifier that is global and persistent, unlike addresses in IPv6 and in the original Internet architecture. (IPv4) addresses serve as *routing tags* and have no end-to-end significance.

A multicast channel as in the EXPRESS single source multicast (SSM) model [4] is identified by a name subordinate to the name of the source host. For example, "chan1.pescadero.stanford.edu" could be the name of a multicast channel with "pescadero.stanford.edu" as the source. (TRIAD only supports the single-source model of multicast.).

3.1 Name-based Transport Connection Setup

In TRIAD, a transport connection is setup by a name lookup and connection setup protocol at the content layer called *Directory Relay Protocol (DRP)* [5]. The client sends a lookup request containing a URL and possibly other information, such as "cookies" to the content layer's directory system. For brevity, we refer to this information generically as the "name", rather than detailing URL and cookies each time. The directory

²Packets are sent to a host interface, not to a host, the same as the original Internet architecture.

system routes the name request through to a content server able to provide this named content, if any, which then either returns the content directly or else sets up transport connection state and returns the associated connection information to the requesting client.

DRP serves as a connection setup protocol for TCP, carrying the same sequence number, port and option information as a TCP SYN packet along with the content identification. The content identification is stored with the connection state and is used to re-bind the connection to new addressing if the connection is failing to get packets through. In the expected case, this rebinding maps to the endpoint that is storing the transport-level state of the connection, allowing the transport-level connection to continue with the new address binding, similar to an ARP-level rebinding. In particular, the rebinding uses the canonical name for the content server so it first attempts to rebind to the same replica. When the original content server is not reachable, a end-to-end name lookup can bind to a new server, restarting the content access from the beginning.

Using DRP, a TRIAD TCP connection to content can be established with a single round-trip from client to server. This contrasts with the two roundtrips commonly required with the current DNS/TCP separation. Moreover, a transparent content cache can intercept a DRP request and service this content request without having to hijack the transport packets because, at the packet-level, the packets are addressed to the transparent cache. This avoids the connection failures that can occur currently when packets are rerouted around a transparent cache. The transparent cache can also receive all the HTTP information in the DRP request, eliminating the need to incur an extra roundtrip to get this information, as with the current TCP/HTTP setup behavior. Also, TRIAD TCP can transparently recover from changes to the addresses used to reach the endpoints, whether caused by intermediate node timeouts or reboots or link failures (assuming an alternate path is available). This rebinding makes the translation in the network effectively *soft state*, preserving one of the key properties of the Internet. UDP-based services are expected to similarly rebind from the name, either periodically or on timeout, in the case of a reliable protocol built on UDP. Finally, DRP carries the client name and identification, allowing the content server to determine this information without callback or reverse name lookup. (The network can validate the DNS-level identification as part of forwarding the request.)

This same name-based connection setup with DRP works for joining a multicast session as well. The name lookup is sent to the source, which returns the required information to join the multicast session, setting up requisite multicast state at the network layer along the path.

For backwards compatibility with current Internet, TRIAD hosts also support (and fail-over to) conventional TCP connection setup when communicating with unmodified IPv4 hosts.

3.2 Name-based Transport Pseudo-Header

In TRIAD, the transport layer checksum covers the name of the source and the name of the destination and *does not* include the packet address. In the case of multicast, the pseudo-header is based on the name of the channel. The addressing allows efficient forwarding of the packet to a destination; the name-based pseudo-header ensures it arrived at the correct destination (even though the names are not present in the header). Thus, a transport connection is between two endpoints identified by name, not address.

A transport-level checksum based on this pseudo-header provides end-to-end reliability because it detects corruption of the packet addresses in transit yet does not need to be modified in transit as part of relaying (or forwarding) the packet even though the packet addresses are modified.

This name-based pseudo-header checksum also allows end-to-end security with NAT because changing the addresses does not require updating the checksum. TRIAD includes a security layer similar to IPsec, WRAPsec, which uses names for identification and the same pseudo-header for integrity check verification (ICV), and provides end-to-end security. Note that dispatch to connection state before validating the checksum is required in TRIAD for both secure and insecure connections, unifying the processing in both cases. With conventional TCP implementations, the checksum is often checked before mapping to the associated TCP connection state.

On connection setup, the local endpoint computes and saves a *precomputed pseudo-header checksum value (PHCV)* based on the name of the source and the destination, similarly at the remote endpoint. The source and destination names are also stored in the connection state record with the PHCV. When a packet is transmitted, the checksum is computed starting with the PHCV, effectively incorporating this name-based pseudo-header into the packet checksum. On reception, the packet is demultiplexed to the TCP connection

state based on the packet addresses and port numbers. This receiving connection state contains the same PHCV, allowing the receiver to (re)compute the packet checksum efficiently. If the computed checksum fails to match that in the packet, the packet is considered corrupted in transmission.

If the packet does not map to a valid connection state, the receiver does a reverse name lookup to determine the source name and looks up the connection by name. If the name lookup fails, the packet is discarded as a corrupted packet. An endpoint may receive a packet for an existing connection that does not match based on addressing (perhaps because of a reboot of an intermediate node). The name-based mapping allows the connection state to be located and the address mapping to be corrected.

To allow connection setups with minimal modifications to current TCP clients, TRIAD-TCP includes a (new) option that carries the PHCV in the SYN packet, rather than relying on DRP. For backward compatibility, TRIAD-TCP can also use the current TCP checksum algorithm for a connection where the source and destination are in the same realm.

TRIAD-TCP provides a negotiated “unreliable” mode, which simply disables retransmissions. This minor extension to TCP as a negotiated option allows applications such as real-time VoIP and video to use TRIAD-TCP and automatically get the rebinding behavior described above (as well as the TCP congestion avoidance mechanisms). With this provision, TCP can replace the wide-area use of UDP in all cases that we are aware of. Then, UDP is only used local to a realm, if at all.

The behavior in TCP of allowing infinite timeouts when the connection is idle is supported in TRIAD by a timestamp on the name stored in the connection record, and rebinding when a connection becomes non-idle if the connection has been idle with no relookup for some excessive period of time.

TRIAD routers include support for WRAMP³, an ICMP-like protocol for sending “destination unreachable” messages, similar to ICMP, thus informing hosts (on a best efforts basis) when the addressing is no longer valid.

With these changes, applications on WRAP-aware hosts using TCP have end-to-end semantics and end-to-end reliability, and are oblivious to loss of intermediate translating state except possibly for the performance impact. These changes to TCP do not change the packet format, are local to the implementation, and allow unmodified hosts to communicate within a realm.

3.3 Aliases and Content Routing

Content routing is supported in TRIAD by associating an *alias name* with the content. The DRP lookup of an alias name is mapped to the associated canonical name that is closest to the requesting client. (An alias can represent a host or a set of hosts.) For example, “yahoo.com” can be an alias for a set of canonical names “yahoo1.com”, “yahoo2.com”, etc. representing the set of content servers for Yahoo. For example, a lookup of “yahoo.com” might select “yahoo3.com”, causing the DRP request to be routed towards this specific server. If this content server does not respond, the first-hop content router can select a different content server. The client can also request exclusion of a non-responsive server in a DRP request.

In this fashion, client requests are routed over the best path to the desired content in the normal case yet can recover from a failing content server. In this sense, TRIAD provides an “anycast” capability at the directory level with network and client control to reselect alternatives based on its direct experience with the chosen server. In contrast, conventional anycast mechanisms at the network layer, such as that provided by IPv6, may repeatedly route anycast packets to a server that is not functioning adequately, based on a higher-level (than network layer) evaluation.

This named-based approach relies on the name mapping being as reliable and as secure as packet forwarding, so applications can use names without loss of reliability or security. It also requires that the directory service have sufficient network routing information to determine the proximity of the interfaces identified by the alias, relative to the requesting client. This is supported in TRIAD by integration of the naming and routing. (Of course, higher-level checksumming, encryption, and authentication can provide additional security and reliability when needed.)

4 Integrated Naming and Routing

In TRIAD content layer, naming and routing are integrated in three senses. First, the router implements name lookup service, rather than a separate server that is off the data path. Second, a name lookup can

³The Wide-area Relay Addressing Management Protocol

return a route or *path specification* to the client. Finally, the routing mechanisms and directory mechanisms are strongly coupled in the router: the routing table maps from name to next-hop, rather than mapping address to next-hop, the routing information identifies endpoints and next-hop nodes by name, and name mapping information is disseminated by routing advertisements throughout the network. Individual hosts can participate in the routing by indicating the names or *content volumes* they support and the “distance” to that content, indicating the cost of reaching that content through them.

In practice, it seems unnecessary and excessive management overhead to actually have every router involved in the directory service⁴. In TRIAD, the key routers involved in the directory service are the firewall/NAT routers between realms and BGP-level routers between autonomous systems. We collectively refer to routers involved in the integrated naming and routing system by the term *content router (CR)*. We use the term *content resolving router (CRR)* for a router that just participates as a DNS resolver. This, an ISP would typically have a small number of CRs, corresponding to its current BGP routers, plus conventional routers within one realm, obvious to the content layer.

Each CR or CRR acts as a name server for each realm to which it is directly connected. For names in the same realm as the requester, the TRIAD directory service behaves exactly the same as current DNS for IPv4 clients making DNS lookup requests. That is, a DNS request with *QTYPE = A* simply returns the IPv4 address of the associated local host, as determined by the local name database. In particular, a content router can use name lookup to locate other CRs in the same ISP (and thus presumably connected to the same ISP realm). For inter-realm lookups, the CR may return a similar local IPv4 address that it translates to the remote destination address or it may return an *address path*, as supported by the WRAP protocol. For now, consider an address path as a sequence of addresses, typically spanning several address realms, designating a loose source route to the packet destination. (Like *split DNS* in NAT, different realms have different paths associated with the same name.) This protocol is described in more detail in Section 5.

Typically, a firewall router participates as a CRR in the ISP realm to which it connects. A name lookup request it receives from a client in its private realm for a name outside of this realm is passed to a content router in the ISP, which returns an address path for a content server to the firewall. The firewall then modifies this address path information by adding its addressing information as appropriate and passes the result back to the client. Thus, each such client of the ISP behaves similar in name lookups and routing participation to that of a conventional NAT router.

4.1 DRP Implementation

A name lookup is performed by recursively relaying the DRP request across CRs from the requestor to an content server for the specified content. At each node, the name request is logically relayed along the path that packets are to take, based on local knowledge of which peer is the “next hop” towards the requested content. After the request reaches an appropriate content server, a response is returned along the reverse path through the CRs, with each one modifying the addressing to finally produce the address path suitable for the requestor to use.

By relaying the name lookup request across the same path as the packets are to flow, any necessary forwarding state can be set up in intermediate CRs. Also, this means that DNS is as available as endpoint connectivity, i.e. if the endpoint is reachable, name lookup to that endpoint works. Moreover, the trust in name lookup matches the trust in delivery because both depend on the same set of network nodes. Also, the name lookup load for a path is imposed just on the routers on that path so upgrading a router on that path in data capacity can also upgrade the name lookup capacity on that path. Moreover, transparent caches, can added to a loaded path, and off-load name lookup and content delivery from subsequent portions of the path. Thus, one can balance data throughput and directory service capacity, similar to how this is handled with file systems. Consequently, in TRIAD, you can rely on names as much as you rely on addresses in the current Internet architecture, and name lookup capacity scales with the Internet.

A CR can also provide reverse (i.e., address-to-name) lookup by forwarding a reverse lookup request along the same path as a packet with the same address.

In a multi-homed realm, such as an enterprise network served by two ISPs, the internal naming and routing selects one of the CRRs for the name lookup based on local routing information. This mechanism is

⁴For instance, within one realm, any of the routers supporting a directory service is reachable by address without the directory service, allowing a client to access this service if there is connectivity to it.

also expected to detect when the selected node fails or becomes disconnected, causing traffic to be rerouted through the other CRR. Because the name is the primary identifier and can be rebound without losing the connection state, the connection can survive this redirection to the other router similar to a connection surviving rerouting in the current Internet. With routing updates significantly damped in the Internet to avoid oscillations, especially at the BGP level, we expect TRIAD name rebinding to provide recovery latency that is comparable, if not superior, to that of the current Internet.

4.2 Name-based Routing

The TRIAD *Name-Based Routing Protocol (NBRP)* [3] performs routing by name with a structure similar to BGP. Just as BGP distributes address prefix reachability information among autonomous systems, NBRP distributes *name suffix* reachability among address realms. Routing information is distributed among CRs and maintained locally with next hops and destinations specified in terms of names and name suffixes. With this step, the name directory and the routing table logically become a single entity, reducing the overall complexity of the CR software⁵.

This *name-based routing* distributes name mapping information to ensure its availability, to distribute the name lookup load, and provide faster name lookup response. Identifying endpoints by name is also necessary because addresses are not unique across the multiple realms in TRIAD. (Intra-realm routing can use existing routing protocols. Intra-realm reliability of name service can be ensured by duplicate servers as now.)

The key challenge with name-based routing is maintaining the routing database efficiency in the presence of names that do not match the routing hierarchy. To reduce routing table size to a feasible level, name-level redirection mechanism is used to handle hosts whose names do not match network topology. For example, all hosts with Harvard names not in the same address realm as the authoritative server for Harvard.EDU would have redirect records at that server. Consequently, the routing database only needs to deal with Harvard.EDU, not hosts subordinate to this domain. Nevertheless, TRIAD can deal with third-level names as necessary, such as europe.ibm.com, japan.ibm.com, etc.

NBRP also supports combining collections of name suffixes that map to the same routing information into *routing aggregates*. For instance, we expect an ISP relay node to group all of the names from its customers into a small number of aggregates. With aggregates, routing updates typically update a small number of aggregates rather than the large number of individual name entries contained in each aggregate. Moreover, all names in a routing aggregate may be treated identically in routing calculations, thus reducing load at CRs. Aggregate membership should be relatively long-lived, so that CRs can amortize the cost of learning the aggregate membership over many routing updates. Section 7 describes measurements of a preliminary evaluation of the benefit of route aggregates.

Although TRIAD name lookups may contain a full URL, cookies and other content level identification, the key information maintained by network nodes is essentially the same as current DNS. Most more detailed content information is maintained at end nodes, and cached in transparent proxy caches.

To keep the number of NBRP neighbors small so that the routing overhead is acceptable, nodes on the interior of a realm can be added that perform only route updates and name lookups but do not otherwise participate in the routing. Current BGP speakers could be upgraded to perform as NBRP speakers as well participating in the routing. The CRR of a typical ISP customer is a degenerate case of this approach.

4.3 Host Advertising, Aliases and Content Routing

A host can advertise an alias associated with its canonical name(s) to the NBRP system together with the hop count cost of accessing the associated content through this host. This cost is in terms of "hop-equivalent" response time units. That is, if the response time cost of going an extra hop is estimated as 2 milliseconds, this advertised cost is k if the server is averaging $2 * k$ milliseconds to respond to a request. The routing system adds this cost to the routing cost of accessing this host as it disseminates the naming and routing information. Explicit advertising of host cost is limited to content servers, a very small fraction of the hosts on the Internet.

A CR receiving advertisements of two or more canonical names with the same alias name groups these canonical names for lookup under this alias. On receipt of a name lookup for the alias, the CR orders

⁵Note that a conventional routing table is a simple directory: It is queried with an IP address to determine the forwarding information. With TRIAD, the equivalent directory in a relay node is queried using the DNS name.

the associated canonical names by proximity (determined from the routing database) to the requestor and forwards the request to the closest proxy.

This approach has several benefits for content routing. Because this information is pushed out to each CR, a client request is immediately routed to a close-by content server that than having to first go to a distant central server. This distribution also avoids excessive load and failure-dependence on a central resource. Moreover, the proximity information is based on server load and availability, not just network access.

4.4 Security

The directory service supports message authentication using public-key and shared-key cryptographic signatures. This allows clients to determine that the answer they get from the directory service is authentic, and allows relay nodes to identify a particular principal associated with a client.

NBRP updates are authenticated by cryptographically signing "delegations" of part of the namespace to a CR's peers, in a manner similar to Secure BGP [18].

Unlike DNS security[7], a single name-to-address mapping cannot be signed by the authoritative server for a name because the address also depends on the intervening CRs. Instead, CRs must establish trust relationships.

5 WRAP and Path-based Addressing

In TRIAD, WRAP, the *Wide-area Relay Addressing Protocol (WRAP)*, is a "shim" protocol that specifies, together with the IP packet source and destination addresses, a *path* to desired content. It carries the transport header and data as its payload, similar to other IP encapsulation protocols.

The WRAP header contains a pair of *Internet Relay Tokens (IRTs)*, the *reverse token* and *forward token*. The forward token represents the path the packet is to take and the reverse token indicates the path the packet has taken to this point. An IRT is a potentially opaque variable-length field that specifies a path from the source to the destination. It may also be simply a sequence of IPv4 addresses.

A WRAP packet is formatted as in Fig. 1 as the payload of an IPv4 packet.

0-7	8-15	15-23	24-31
protocol	length	offset	reserved
reverseToken			
forwardToken			
data			

Figure 1: WRAP Packet Format

The protocol field specifies the higher-layer protocol in the "data" field using the same types as for IP, e.g 6 for TCP. The length field is the number of 32-bit *components* in the header. Thus, the WRAP header length in octets is $4 + \text{length} * 4$. The components specify the reverse and forward tokens. The *offset* field is an offset into the list of components where the *forwardToken* starts, with 0 referring to a null *reverseToken*, so the forward token starts in the first 32-bit field. The *forwardToken* is used by the node addressed in the IP packet destination address to determine how to relay the packet to its destination. The *reverseToken* is a value used by the node addressed by the IP packet source address to refer to where the packet came from.

The data field contains a TCP, UDP, or other transport protocol packet.

A WRAP source sends packets to a destination by forming an IPv4 packet with the IP destination address set to the address of the next relay, the WRAP header containing the IRT in the *forwardToken* of the destination relative to this relay node, and a null *reverseToken*. Thus, the length field is the length of the *forwardToken* and the *offset* field is 0. (The *offset* value is also the length of the *reverseToken*.)

A node, on receiving an WRAP packet:

1. maps the (SA,DA) of the packet to a *virtual interface (VI)* that represents the local endpoint of the realm "channel" on which the packet arrived. It maps the next *k* 32-bit components of *forwardToken*

field (assuming offset is less than length) to a corresponding relay entry in a relay table associated with this VI.

2. determines from this entry the next IP source address (the "egress" interface), the next relay's IP address, the new forward token, and the rewrite of the reverse token to perform.
3. forwards the modified packet to the next relay node, with the IPv4 destination address as that of this next relay and the IP source address determined above, and increments the offset field.

Each relay node thus "consumes" one or more 32-bit components from the forwardToken and adds an equal number of components to the reverseToken. (However, both these fields may be translated according to the information in the relay node's lookup table.) The reverse token, when component-reversed, must be recognized by this node when used as a forward token, to send packets back toward the source.

Normally, an node just rewrites the first forward token component and increments the offset by 1. In the simplest case, the rewritten component is just encodes the IP source address of the incoming packet (that is, the last relay point.) This restricted form of relaying, though less general than WRAP allows, is more amenable to hardware implementation, because less of the packet needs to be rewritten.

If the node does not recognize the forward token, it drops the packet and may send a WRAMP message back to the previous node. The relaying state may include filters on sources from which to accept packets and destinations allowed for given sources.

The receiver of a WRAP packet is a node that receives the packet with a null forwardToken, or receives a packet with a multicast destination and subscribes to that multicast source.

The actual source of the packet is identified by the reverseToken and the IP source address. The receiver can contact this previous relay identified by the IP source address to do a reverse name lookup on this IRT to determine the name of the actual source.

With WRAP, a packet is reassembled from fragments at each intermediate relay node, because each is a destination from the IP standpoint. This feature reduces the risk of carrying packet fragments all the way to the destination only to discover some fragment is missing. Each WRAP node sets the TTL of a WRAP packet according to its estimate of hops to the next relay. Packets cannot loop at the WRAP level because some non-zero portion of the WRAP IRT is consumed at each relay node.

Backbone or wide-area ISPs can connect at peering points, the same as today, but with high-speed relay routers at these points. At this level, the firewall or border router is extended to act as a TRIAD relay between realms, translating packet addresses as it relays packets between the realms that it interconnects.

Between the IP addressed nodes, a packet is routed by the normal IPv4 routing protocols used within the realm. Thus, WRAP is similar to loose source routing with the relay nodes as the designated nodes on the path it is to follow.

Within a realm, the operation of naming, addressing and routing operates the same as currently with IPv4. Thus, there are no host or router changes required. A packet that does not travel outside of a single address realm can omit the WRAP header entirely.

5.1 WRAP Example with Multiple NAT Realms

Fig. 2 illustrates the operation of TRIAD between realms with two hosts, src.Harvard.EDU and dst.Ietf.ORG, assuming Harvard.EDU and Ietf.ORG are two separate realms connected via a single intermediate realm, the "external" Internet. (For simplicity, we illustrate just with DNS names, not full URLs.) For src to send to dst, the name lookup of dst.Ietf.ORG is handled by the relay node relay.Harvard.EDU for this realm, with internal IPv4 address RA1 and external IPv4 address RA1'. This relay determines the appropriate next relay from its directory mapping of Ietf.ORG and then communicates the name lookup across the Internet to the relay. Ietf.ORG, the relay for the Ietf.ORG realm. (This relay has internal IPv4 address RA2 and external IPv4 address RA2'.) In response to this query, relay.Ietf.ORG communicates with dst to set up connection state and then returns to relay.Harvard.EDU an IRT f' that designates dst relative to RA2' and the associated transport connection information. Then, relay.Harvard.EDU returns an IRT f to src which designates dst.Ietf.ORG relative to RA1, creating any state it needs to map f to f', passing the transport connection information through.

Then, src sends the first data packet over this connection as an IPv4 packet addressed to RA1 with f stored in the WRAP header. On reception, RA1 translates f into f' and transmits the packet with

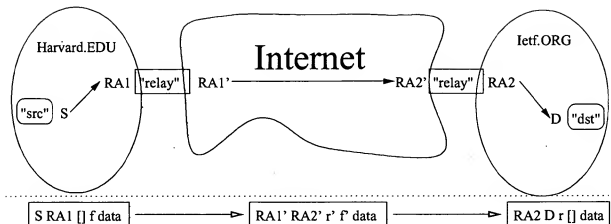


Figure 2: Inter-realm packet transmission in TRIAD: The host named "src" with IPv4 address S in realm Harvard.EDU sends to the host named "dst" and IPv4 address D in realm Ictf.ORG. The packets below the dotted line indicate how the IP and WRAP headers changes as it crosses the three realms, with the header listed as source address, destination address, reverse IRT, and forward IRT.

destination address $RA2'$ and source address $RA1'$, as shown in the middle packet in the figure. The WRAP header also contains the reverse IRT r' , which indicates the source of the packet relative to $RA1'$.

On reception at $RA2'$, the reverse IRT in the packet is translated to a new value r which represents src.Harvard.EDU relative to $RA2$. This relay then transmits the packet with an empty forward IRT, IPv4 destination address D and source address $RA2$, as shown in the rightmost packet in the figure.

Thus, dst receives the packet as a normal IPv4 packet sent by its relay $RA2$ but also containing an IRT that identifies the actual source of the packet relative to its relay. The packet is then passed up to the next higher layer processing, such as TCP or UDP.

The destination can respond to a packet directly by "reversing" the IRT and sending the packet to the local relay with this reversed IRT. This causes the packet to return along the reverse of the relay path on which the original packet was received. Alternatively, the destination can perform a lookup on the source name to get a separate IRT and RA to reach this host. This alternative is more flexible, allowing for asymmetric routing at the cost of an extra name lookup.

An address realm at the leaf level may correspond to an enterprise or university network, a military installation, or much smaller units like a collection of autonomous sensors or a home network or even a set of virtual hosts on a single physical machine. Higher-level address realms correspond to local and global Internet service providers (ISPs).

The IRT and the relay address are local in scope and transient. That is, the IRT is only meaningful relative to the relay and realm and is only guaranteed to be *T-stable*: it does not go from one valid association with a relay to another in less than time T , where T is typically hours. In particular, it can become invalid at any time but can only be reassigned to another use after time T . Thus, passing an IP address or an IRT in the data portion of a packet to the other endpoint is meaningless in general.

A WRAP proxy, referred to as a WRAPID gateway (see Section 8), allows existing IPv4 hosts to interact with WRAP-enabled hosts and servers without any modification. A WRAPID gateway is just an extended NAT-capable router or firewall which is able to WRAP and unWRAP packets going through it, as appropriate.

5.2 Transparent and Opaque Relaying

WRAP allows the relaying to be *transparent* in the sense that each IRT is simply a sequence of IPv4 addresses designating relay nodes and endpoints, an *Internet Relay Path (IRP)*. The IRT can also be *opaque* so that a holder of the IRT cannot determine the relay path nor can it forge a valid IRT.

Using a transparent IRT, the relay is stateless in the sense that the relaying only relies on routing/directory state and configuration state; it does not require state to be created on name lookups. In this mode of

operation, the relay node is statically configured with an IPv4 address for each of the other realms it connects to, so that an address uniquely identifies which direction to relay a packet (and a particular “egress” address in that realm.) Upon receipt of a WRAP packet, the relay replaces the IP destination with the first forward token component, uses the egress address as the new IP source, and places the old IP source as the last component in the reverse token, as described in the earlier example. This is the simplest possible relaying action, requiring only 4 words in the packet headers to be modified.

To make the WRAP addressing opaque to an observer, the relay node can choose to put a random value in the IRT and translate it to/from IPv4 addresses using the relay table described earlier. This opaque form prevents a upstream source from fabricating IRTs, forcing it to rely instead on the directory service to supply IRTs. In particular, an ISP can retain control of routing, preventing customers from using unauthorized routes. It also prevents a third-party observer from determining protected information from addresses in the packets.

An IRT must have the *reversibility property*, namely that the component-wise reversal of the received IRT provides an IRT that can be used to send a packet back to the source of the packet using the relay from which the original packet was received.

An IRT normally also has the *concatenation property*, i.e. if the IRT to a relay is X from a host H and Y is the IRT to a destination D relative to this relay, then XY is an IRT to destination D from host H . The directory indicates whether the returned IRT supports concatenation or not.

5.3 Multicast

WRAP supports the EXPRESS [4] single-source model of multicast. Multi-source multicast applications can be supported by relaying the multicast through a node that is a source of an existing relay multicast channel, similar to the rendezvous point in PIM-SM, but performed at the WRAP or the application layer.

A subscriber joins a multicast channel by specifying its name in a DRP lookup, which returns the required multicast channel addressing information.

A multicast WRAP header contains the same multicast address G repeated r times, where r represents the maximum number of relay hops in the multicast tree; this allows multicast WRAP relaying to be performed identically to unicast WRAP relaying. As multicast packets are relayed, group addresses may be translated so that the (S,G) pair upon which IPv4 routers do multicast delivery is unique. However, a single intra-realm channel can be reused within a realm to deliver multiple inter-realm channels.

5.4 Content Routing and WRAP

WRAP allows the directory to dictate a path for packets to take from the client to receive the delivery performance it has determined, rather than leaving the client packets to be routed by a separate mechanism, as occurs now.

WRAP also allows the directory to return a path specific to a requesting client, rather than an address that is generally common for all clients.

Finally, WRAP allows a server behind a NAT box to be addressed without creating translation state in intermediate nodes.

5.5 Secure Communication

TRIAD includes a secure communication facility similar to IPsec, i.e. end-to-end at the (inter)network layer. It differs primarily in working in the presence of NAT or WRAP translation, because the Integrity Check Value (ICV) does not include the packet addressing information, similar to the TRIAD-TCP pseudo header. Here, the principal associated with the connection can be identified by name.

6 Additional TRIAD Benefits

TRIAD provides benefits in mobility, VPNs, policy-based routing and extended reverse path forwarding checking, in addition to its support for content routing, NAT and scalable addressing, as outlined below.

6.1 Mobility

For mobile operation in TRIAD, a host visiting a guest network receives a temporary visitor name in that network (in a DNS domain of the visited network) which allows it to then communicate with the rest of the Internet. If the host needs to be reachable or authenticated as its normal DNS name, it gets its home directory service to insert a redirect this name to its current temporary visitor name in the guest network. It

also notifies the guest network of its home identity, based on name. When another host attempts to contact the visitor by its normal name, the home directory provides a redirect to the temporary visitor name, causing the other host to then contact the mobile host directly in the guest network.

When the mobile host moves, transport connections can continue to function even though its address may change. The mobile host simply acquires a guest name in the new network and registers its real name with the guest network and its guest name with its home network. The transport connections simply rebind based on the name identification, the same as required when NAT translation state was lost or changed in the network. For real-time hand-off between guest networks, the mobile host can request that a relay node in the old network forward its packets (using encapsulation) to the new guest network for some limited period of time. This forwarding is canceled before the relay node reuses the address and name that was used by this guest host (allowing the relay node to use common state and time-out mechanisms to control the forwarding and the reuse). A reverse name lookup can also return the "real" name that redirects to this (temporary) name, providing what might be called *reverse aliasing*. A lookup on this real name is used to validate this reverse alias.

With this approach, the key mechanism to support mobility is the adding and removing of redirects in the home directory of the mobile host and the registering of the mobile host in the guest realm. The guest network simply needs to allocate and reclaim temporary addresses and names the same as supported by current DHCP services. It does not require routing all packets to a mobile host through the home gateway for the mobile host or encapsulating traffic to and from the mobile host, as mobile IP proposals imply.

6.2 Virtual Private Networks (VPN)

Using WRAP, an ISP can provide a *Virtual Private Network (VPN)* service by creating for each enterprise a secure (virtual) transit realm that connects to each of its enterprise sites. The enterprise directory and routing system only needs to deal with the topology of the enterprise network with this "virtual" realm directly interconnecting all the sites. The different sites of the VPN can even have overlapping IPv4 address assignments (typically 10.X.X.X) yet still communicate directly without renumbering.

The ISP implements this virtual realm simply by providing routing and secure communication between each site. In this case, the overhead from WRAP is 12 bytes. As an optimization, the ISP can provide at each site a relay node address for each other remote site in the VPN so packets can be addressed as though each site was directly connected through a relay to each other site, reducing the header overhead to 8 bytes. Thus, WRAP can also obviate the need to deploy MPLS [14].

6.3 Policy-based Routing

WRAP supports policy-based routing across multiple realms⁶ by using the WRAP path-based addressing to direct packets through certain relay nodes and to avoid others, with the directory mapping particular names to these policies. For instance, a special name in the name lookup can provide a special IRT to a destination that directs packets over a more secure ISP network to a particular destination rather than using a cheaper but less secure route. The ISP directory service can also provide different IRTs based on the class of service that the requesting customer is paying for. It is similar in this sense to source routing and tunneling, but with the key differences discussed in Section 10.

This routing control can also be used for traffic engineering.

6.4 Extended Forwarding Path Check

WRAP supports an *extended forwarding path (EFP)* check based on the WRAP header indicating the (relay) path it took to the receiver, not just the port that the packet arrives on. The receiver can verify that the packet was received from trusted relay node based on the IP source address, only trusting the local network realm to prohibit source spoofing. It can further rely on the relay node to only accept packets from trusted relay nodes in other realms. With this constraints, a reverse path name lookup reliably yields the name of the source, at least to within some originating domain.

With this approach, the true source of the packet is explicitly specified in the packet, up to the trustworthiness of the relay nodes. The conventional reverse path forwarding check is only used within a local realm to prevent local source spoofing. Thus, a receiver or relay can check whether the relays that the packet took

⁶Each realm can support its own local policy-based routing.

are trusted and accepted, independent of whether it would forward a packet to the source of this packet back along the same path.

Unlike conventional source routing, WRAP operates with strict reverse path forwarding (RPF) checking in place and does not allow source spoofing attacks.

RPF checks at the IP and WRAP levels are important because so-called source spoofing is the basis for many denial of service and security attacks. These attacks and various forms of network device failures and misconfiguration are a growing concern with the scaling of the Internet. A key part of handling these problems is having a reliable means of verifying the true packet source. Encryption techniques providing authentication and confidentiality can, by their cost in processing, actually make denial-of-service a bigger problem. That is, the increased time to decrypt a packet with secure communication, only to discover it is a bogus packet, means a node loses more resources in an encrypted denial-of-service attack than with plaintext messages. (Providing wire-speed hardware-supported encryption addresses this problem in part, but is an expensive solution for low-end systems and generally does not deal with setup processing, such as PKE-based authentication on connection setup.) For mission-critical applications, denial-of-service may be as damaging an attack as any of the other possible security attacks. We view ERPF allowed by WRAP as an important feature for scaling anti-source spoofing and dealing with these DoS concerns.

7 Implementation and Evaluation

We have developed a prototype implementation of the extended directory and routing service required in TRIAD. The key issues are the client name lookup performance and the directory/routing storage and maintenance overhead.

Regarding name lookup, we expect most environments to use transparent IRTs, which have the *concatenation property* mentioned in Section 5. Thus, the caching of names behaves the same as with current DNS because a name server can lookup the address to a server once, then send name requests using the relay fast path to this server rather than through the name service on each intervening relay node. The concatenation property also allows addresses looked up for one client to be used for another client on the same "side" of a relay node. Caching of names thus behaves the same as with current DNS.

Furthermore, content lookups would be typically handled by a content cache on the path to a primary content server, providing faster client service than the current Internet and keeping the name lookup (or content routing) local to this portion of the Internet close to the client.

Opauing the IRTs can defeat caching, particularly if the returned IRT encodes source dependencies, but the cost is low compared to the other overheads with secure connection setup.

On a name cache miss, in TRIAD, the name lookup may proceed through several relay nodes, causing a full name lookup at each relay node. In contrast, a conventional DNS name cache miss (within an enterprise) causes a DNS request to be sent to a root name server. Thus, TRIAD may use more cycles in total, summed across several relay nodes, but it distributes this load over the relay nodes on the path of communication. In contrast, DNS incurs fewer total lookup cycles but concentrates the demand on the smaller number of root servers.

The number of name suffixes which must be searched is large, but not unacceptably so. There are currently 1.7 million second-level names in use world-wide, e.g. Harvard.EDU, Ietf.ORG, etc. (This number closely matches the number of suffixes obtained from the experiment explained below.) Assuming 64 bytes of space per entry (including hash indexes, etc.), storing the whole name database would cost 128 megabytes, an insignificant amount of disk space, even if the number was to be 10 times as much by the time TRIAD was deployed. NBRP table lookup is not on the packet forwarding fast path, unlike IP routing, so time spent searching the table is typically only paid during connection setup rather than per-packet. Note also that a name lookup already encounters the cost of searching through a database of this size in conventional DNS.

In sum, we expect TRIAD name lookup to have comparable performance and scaling as current DNS, differing primarily for portions of the Internet configured for greater security requirements than supported by current DNS.

Considering the directory and routing overhead, at the ISP level, the name aggregation generally closely matches the address and routing aggregation. For example, Harvard.EDU corresponds to a small number of IP address ranges that further correspond to a small number of routes. This strong correspondence means the aggregation feasible with routing table entries is largely intact in going to name-based routing and

directory services. Conversely, organizations with large numbers of hosts scattered throughout the Internet are uncommon.

7.1 Expected NBRP Performance

To evaluate the expected performance of name-based routing in the current Internet, we processed a comprehensive list of address-to-name mappings in the Domain Name System[19] and BGP table dumps from the MAE-East exchange point[20] by the following algorithm, making the assumption that address realm boundaries roughly correspond to current BGP autonomous system boundaries:

1. Each address range from the BGP table is matched with the DNS zones represented. (If fewer than *site.threshold* hosts in a range belong to an existing zone, they are removed from the table completely and assumed to be handled with the redirection mechanism.)
2. Names whose associated routing information is made redundant by a superzone are also removed.
3. Aggregates are created for any set of names larger than *aggregate.threshold* that have identical routing information (i.e., all known routes were identical, not just the preferred route.)

The resulting aggregates match those expected to be generated in a TRIAD relay node.

One representative set of results is shown in Table 1. These numbers indicate that NBRP results in a

<i>site.threshold</i>	Affixes (1000s)	<i>aggregate.threshold</i>			
		3	5	10	20
2	1727	19.5 (6.7)	20.1 (5.6)	25.7 (4.4)	37.0 (3.4)
3	1692	14.9 (5.9)	16.1 (5.0)	20.6 (4.0)	30.1 (3.2)
10	1679	14.8 (5.9)	16.0 (5.0)	20.6 (4.0)	30.3 (3.2)
original BGP	68.2	11.8			

Table 1: Number of routes (and aggregates) in thousands for different site and aggregate threshold values. With a site threshold of 10 and an aggregate threshold of 3, NBRP produces approximately 14,800 routing table entries (and 5,900 aggregates) which improves significantly on the original BGP number of 68,200 routing table entries.

set of destinations (and thus update frequency) comparable to BGP; higher-level aggregation may be able to reduce this yet further without resorting to renumbering or renaming.

BGP does have a limited mechanism for aggregation: a single route update may include several address prefixes. It is not clear the extent to which BGP software makes use of this to optimize update calculations: there is no requirement that advertisements keep these address prefixes together, and the address ranges must appear separately in the IP routing table. The entry in Table 1 corresponding to "original BGP" with an aggregation threshold of 3 indicating 11,800 entries indicates the best possible number of routes with BGP aggregation.

Addition of a new name is common, unlike addition of new BGP prefixes, and this name information must propagate to all relay nodes. However, addition of new names is done on human time scales; during the recent past, third-level domain names have been added at about 12 per minute. To put this in perspective, a backbone router may receive more than 2,000 routing updates per minute. Also, the actual level of routing updates necessary for new names is lower because changes to aggregates can be "batched" to reflect many new names with one update.

7.2 WRAP Implementation and Performance

WRAP incurs a low space and time overhead for communication on average because communication within a realm just uses the conventional IPv4 header. Given that most communication is local and the current Internet with NAT boxes is effectively at most 3 relays to anywhere, the packet header overhead on average is expected to be significantly less with WRAP than with IPv6⁷.

⁷One could argue that the Internet does not actually need more global addresses, by relying on efficient allocation and NAT, given only about 1 percent of the IPv4 addresses are actually in use. However, WRAP is still beneficial for other reasons, such as connecting private address domains, VPNs and content routing.

This header overhead is significant because most packets are small and per-packet processing is a significant cost with small packets. This optimized local communication also suits small embedded systems, many of which use or will use limited bandwidth wireless communication. Moreover, it is readily hardware implementable because of the size of relay address to lookup can be fixed size.

In comparison to conventional forwarding, relaying requires an additional lookup of the next forwarding component in the context of the virtual interface to which the packet is mapped with the (SA,DA) lookup. A hardware implementation may add an additional lookup resource to handle this or simply perform two lookups on the same memory, depending on the speed of this memory and the forwarding performance requirements. We expect initial hardware implementations may restrict the *offset* they support, throwing packets with larger values to software.

Multicast relaying uses additional state in the forwarding path but is the same implementation and performance.

Our Linux implementation of WRAP added about 1,500 lines of code as a kernel module and incurred an extra 2.2 microsecond overhead (or 2.6 percent) for relaying compared to conventional IP forwarding.⁸ Thus, the complexity is minimal for either software or hardware, and the software performance overhead is minimal, and comparable to that required for NAT forwarding.

8 WRAPID Gateways

A WRAPID (WRAP-to-IP-Domain) gateway allows existing IPv4 end hosts to operate with TRIAD without modifications to their software. WRAPID provides translation between IPv4 addresses and WRAP addressing similar to the IPv4 to IPv4 translation provided by NAT boxes.

A WRAPID gateway handles outgoing conventional DNS lookups, performing a DRP lookup using just the DNS name. The WRAPID gateway allocates an IPv4 address for this remote server and sets up a mapping to the appropriate WRAP header. This header may map directly to the remote host or to a WRAPID gateway that serves that host. When an (IPv4) packet is sent to this allocated address, the WRAPID gateway translates the packet to a WRAP packet with the appropriate header and forwards it onwards. On receiving a WRAP packet from an external host, the WRAPID gateway translates the packet to a simple IPv4 packet with the IP source appearing as this locally allocated address.

The WRAPID gateway also handles incoming DRP requests, performing the name lookup internally, and then requesting a connection setup at the host, using the URL information in the DRP request and sends an HTTP request to the client, if that information is present in the DRP request. It then returns this connection information and splices the client connection into the connection it has established to the server.

The WRAPID gateway can also implement WRAPsec, providing secure communication to the other WRAP endpoint, either a WRAP-enabled host or another WRAPID gateway.

The WRAPID gateway allows WRAP to be deployed incrementally. In particular, one can have hosts on the same subnet being WRAP-enabled while others are not, yet still able to communicate with each other as well as hosts in other address realms. The optimization of eliminating the WRAP header when communicating within the same address realm means that a WRAP-enabled host never sends WRAP packets to other hosts in the same realm, so there is no need to discriminate between these hosts as part of local communication. Only the directory service interfacing to the rest of the Internet needs to distinguish. However, a full TRIAD implementation (with the attendant host changes) is required to provide end-to-end security and reliability.

9 TRIAD Deployment

TRIAD has a simple deployment path, based on user need, allowing TRIAD to be realized as an incremental evolution of the current Internet.

At the content layer, DRP and NBRP can be implemented in firewalls and inter-realm routers with the additional capability to fail over to using the current Domain Name System, etc. to implement the functionality in regions of the network that do not support TRIAD. Similarly, content resolvers can make use of the existing naming infrastructure to locate other TRIAD gateways rather than participating in a dynamic routing protocol. We expect deployment to occur mainly at the edges of the network, and thus cannot depend on ISPs providing new infrastructure. Such a scenario could lead to an topology that would

⁸The test machine was a 333 MHz Celeron with 128 MB of RAM, running Linux 2.2.13.

have many thousands of realms peering in the "global" Internet, but there is little need for NBRP in such an environment. The amount of topological change in such a situation is small, and multihomed sites can easily list all their gateways as NS records rather than constantly updating the DNS. Later, these CRRs can be upgraded to content routers as ISPs begin offering NBRP.

Deploying TRIAD for NAT is also compelling. Consider as an extreme example a foreign country with limited IPv4 addresses such as Thailand. The limitations of conventional NAT make it questionable as a solution to providing more addresses yet moving to IPv6 does not make sense either, given the limited deployment of IPv6, the limited product support, and the need to communicate with the IPv4 portion of the Internet. However, with TRIAD, each such country can install a WRAP relay router that interfaces to the Internet. Attached to this top-level relay are one or more WRAPID gateways that include conventional NAT capability. The conventional NAT capability allows these hosts to communicate with the existing conventional IPv4 Internet. Each ISP, country or even organization that adopts TRIAD is able to communicate with other organizations using TRIAD *without* consuming any of its global IPv4 addresses⁹. For instance, if Thailand and Indonesia both adopt TRIAD, they then have virtually unlimited addresses internally and between themselves, and are only constrained on the number of addresses they have available to communicate with the current Internet. (This is actually the same situation as if they had internally converted to IPv6, given they would still have to communicate with the rest of the planet using IPv4. But, with IPv6, they would also have to upgrade all their existing hosts and networking infrastructure.)

Thus, each organization is motivated to adopt TRIAD because it allows them to communicate with other TRIAD organizations without using their limited global IPv4 addresses, and because it makes it easier for other TRIAD users to communicate with them. So, those organizations that are currently short of addresses are motivated to move to TRIAD and those that are not are still motivated if they are interested in having the former communicate with them. Given that most of the major web sites are in the United States, and the U.S. companies have been in the lead to build Web-based operations, there would be considerable commercial motivation to support TRIAD in the American web sites once foreign companies were using TRIAD among themselves.

This initial deployment requires no real changes to end hosts and no change to the basic IPv4 routers and switches constituting the infrastructure of the leaf and backbone networks. It only requires the deployment of content routers and WRAPID gateways, but these are modest extensions of the current NAT-enabled routers. Here, we assume that end-user applications have been or will be modified in any case to deal with the lack of meaning of addresses across NAT boundaries.

Once WRAP is deployed to some degree in the Internet, first host implementations are expected to arise with large-scale servers where eliminating the extra overhead, delay and point of failure of a WRAPID gateway may be warranted. Making an externally accessed server WRAP-enabled also eliminates the server use of an externally visible (IPv4) address which, with an active server, would be essentially allocated indefinitely to this server. During this transition, conventional IPv4 hosts and TRIAD-aware hosts can easily and efficiently co-exist in the same address realm. Given that WRAP appears relatively straightforward to implement, the main delay in getting all hosts upgraded to WRAP is expected to be the basic inertia in getting changes into commercial software and getting administrators of systems to upgrade their software. Hosts that need end-to-end security and reliability are also motivated to upgrade to native WRAP.

Consequently, TRIAD is readily deployable incrementally. There is no need to change the network infrastructure within an address realm or to change backbone routers and management. The boundary (NAT) routers are upgraded to support TRIAD and then the hosts can then be individually upgraded to use WRAP natively.

10 Related Work

The original Internet directory service was supplied by a "hosts.txt" file that listed all hosts in the Internet. As the Internet grew, this approach was replaced by DNS [6] in 1985. Subsequent work on so-called network directories such as X.500 have suffered from misguided objectives of supporting naming of other types of objects such as mailboxes and providing more flexible ways of specifying identification, such as lists of attributes.

⁹This assumes the gateway already has one such address if the WRAP relays communicate over the existing wide-area IPv4 infrastructure.

TRIAD draws on the direction established in the web of treating both host name and file name as part of the path name to content, and unifies the handling of these two portions of the URL. It further builds on the *decentralized naming* approach advocated from experience in the V distributed system [21], and effectively implemented in file systems, which can be summarized as: Names are external identification of objects and a server names what the objects it implements — nothing more and nothing less”.

Current wide-area content routing depends on HTTP or DNS-level redirect. For instance, Cisco’s Distributed Director (DD) redirects a name lookup from the main site to a replica site closer to requesting client address, based on responses from a set of participating routers running an agent protocol, supporting DD. Unfortunately, the client incurs the response time penalty of accessing this main site DD before being directed to the closer site. Proprietary schemes by Akamai, Sightpath, Arrowpoint and others appear to work similarly.

Web caching was introduced by the Harvest project, spawning a whole industry of vendors. Transparent caching evolved to eliminate the need for explicit configuration and the difficulty of configuring hierarchical caches.

Protocols such as ICAP, Cisco’s WCCP and Arrowpoint/Cisco’s CAPP define communication between web caches, web caches and routers, and other content distribution devices. To date, these and other proprietary protocols have not been architected into a coherent Internet architecture.

The XNS [23] Sequenced Packet Protocol (SPP) used a separate connection setup protocol, similar to the separation between DRP and TRIAD TCP we are proposing.

NAT was introduced into the Internet in Jacobson’s insightful early proposal [11] although the same techniques appear in some earlier distributed systems work [10]. Since 1992, various RFC’s [12] have clarified the use of NAT, provided for private addresses [13] and clarified the terminology, use and problems [17]. Industry has deployed a variety of products supporting network address translation, including firewalls, routers and server load balancing switches. More recently, work on IPsec and others have recognized the problems with basing identity on IP addresses and the conflict of end-to-end security with the increasing deployment of NAT.

RSIP [15] is an approach to dealing with NAT, where a host in a NAT realm explicitly obtains an external IP address, tunnels packets through the NAT gateway using this external IP address and thus can use IPsec and other protocols without requiring NAT translation. However, RSIP requires host modification to operate in this mode and it does not increase the number of external IP addresses. With all the extra benefits that TRIAD provides, it seems more effective and lower risk to modify the hosts to support native TRIAD.

The path-based WRAP relay model of addressing as an extension of the basic forwarding level of conventional routers is similar in some respects to the Sirpent [1] form of loose source routing except WRAP is designed to work with IPv4. WRAP relaying is similar to loose source routing except the packet is forwarded at each relay with the source IP address that of the relay, not the original source address. Moreover, each specified address may be in a separate address realm, with translation between address realms occurring at each realm boundary. Source routing provides a source-controlled path but does not cross realms and does not change the source address on each hop, as is required for inter-realm communication. TRIAD path-based addressing could be provided as a new IP-level option. However, using a separate shim header seems preferable because of the inefficiency of routers handling packets with IP options, given that some options need to be handled by each router and others only need to be handled by the IP-addressed endpoint. Moreover, WRAP makes it easier for a hardware implementation to determine the offset of the transport header, which is important for layer 4 access control lists. WRAP is similar in this respect to IPv6 header extensions.

IP tunneling has been used to effectively extend addressing by tunneling from one realm to another. However, tunneling makes layer 4 filtering harder because, with multi-hop tunneling, the location of the layer 4 header involves parsing each encapsulation. Also, unlike WRAP, the path the packet takes is lost with tunneling. Moreover, tunneling incurs greater overhead than WRAP and requires that the source know the path. Moreover, the packet size does not change with WRAP, unlike encapsulation and de-encapsulation that occurs with tunneling.

MPLS [14] provides tagging of packets similar to WRAP, but below the IP level. MPLS does not provide more addresses beyond that provided by NAT, unlike WRAP. On the other hand, WRAP can be used intra-

realm and inter-realm for traffic engineering and VPNs, reducing, if not eliminating, the need for MPLS¹⁰. MPLS also requires special support in the forwarding path of *all* routers on the path, whereas WRAP and TRIAD only require support at the border or relay nodes. MPLS also requires a new mechanism for distributing tags. MPLS does not save the path a packet followed either. While the WRAP header does impose a higher overhead than an MPLS tag, it is less than IPv6 and less than conventional IPv4 tunneling, especially with multi-path tunnels. Thus, IP4 plus MPLS is not a solution to scaling and IPv6 plus MPLS carries all the disadvantages of both. Both WRAP and MPLS make the offset of the TCP/UDP ports variable within the packet, affecting the design of access control filters on packets. However, with the length field in the WRAP header at a fixed offset, it is straightforward for even a hardware implementation to determine the actual offset of layer 4 ports, as required for access control processing. Moreover, in initial deployment, we expect that firewalls may simply restrict WRAP packets to specific WRAP-enabled hosts, such as WRAPID gateways, which can filter further as needed.

Recent IETF work has promoted "transparency" as an important property to achieve in the Internet, defined as "a single universal logical addressing scheme and the mechanisms by which packets may flow from source to destination essentially unaltered" [16]. We view that TRIAD provides transparency under this definition, viewing the "logical addressing scheme" to be DNS naming and the transmission of data without changing the data or its checksum as "essentially unaltered". The changing of the addressing in the packet is not real alteration because corruption by intermediate points is as detectable as with conventional end-to-end delivery.

The restriction of IP multicast to single-source was proposed in EXPRESS [4]. This single-source multicast approach is now being deployed.

11 Concluding Remarks

TRIAD is a promising candidate for the next generation Internet architecture. It addresses the key problem of scaling content distribution by defining a *content layer* that directly supports efficient content routing, transparent caching and content transformation. It further supports network address translation while providing end-to-end semantics, reliability and extensible addressing. Besides these benefits, TRIAD also provides a significantly improved directory service as well as innovative approaches to mobility, virtual private networks, policy-based routing and source spoofing. Compared to IPv6, TRIAD is more backwards compatible, more deployable, more efficient and more secure while providing the same end-to-end semantics and recovery relative to network failures.

TRIAD, as the name suggests, is based on three key ideas. First, TRIAD places the external character-string name as the means of identification of endpoints, relegating the packet address to the role of a transient routing tag. In doing so, it makes network naming consistent with that used in file systems, where similar hierarchical names map to files, and internal system identifiers or handles are generated and used for efficiently in the file access operations. In fact, one can recognize both network and file-level names designate content or state, and see their combined usage in web URLs.

Second, TRIAD integrates naming, routing and connection setup into a content layer, recognizing name lookup needs routing information to locate the closest replica to the requesting client. It also needs the same reliability, security and performance as routing. The integration of transport-layer connection setup with the name lookup allows the name lookup to be end-to-end while reducing the roundtrip delays for content access. As the bandwidth of the Internet increases to multi-gigabit rates, roundtrip times are becoming the dominant client performance issue. This integration also facilitates better failure handling between the directory and transport layer.

Finally, TRIAD extends packet addressing with the ability to specify a variable-length path to the destination in a shim protocol called WRAP, allowing the directory service to control the path a packet takes. This path addressing also provides extensible addressability between address realms, efficient virtual private networking and scalable anti-source spoofing. The simplicity of WRAP makes it feasible to implement in hardware in the next generation of switch/routers, allowing wire speed relaying, even at the highest performance levels. Moreover, intra-realm communication can optimize out WRAP, incurring the same packet overhead in size and processing as IPv4.

¹⁰One of the original motivations for MPLS, efficient IP forwarding, has been eliminated by the advent of wire-speed hardware IPv4 forwarding engines.

Our current work is focused on designing and implementing the detailed protocols required by TRIAD and performing further evaluation and study to support this direction. Specifically, we are performing much more comprehensive studies of the large-scale behavior of name-based TCP and routing through simulation. The ability of name rebinding to handle routing topology changes and the effects of route aggregation need to be clearly demonstrated before TRIAD can achieve wide-scale deployment. However, we have confidence in TRIAD's scalability, since the dynamics of naming and routing are similar to what already exists in the IPv4 Internet.

We believe the primary competition to TRIAD at this stage is the continued *ad hoc* deployment short-term fixes and specialized mechanisms, including the proprietary approaches that have arisen for dealing with content distribution. Continued growth of the Internet without a guiding architecture risks increasing entropy as a result of these fixes, detracting overall from its future reliability, availability and security. We see TRIAD as an alternative to this unfortunate direction.

References

- [1] David R. Cheriton, Sirpent, Proc. SigComm'89, 1989.
- [2] David R. Cheriton and Chetan Rai, Wide-area Relay Addressing Protocol (WRAP), in preparation, 1999.
- [3] Mark Gritter and David Cheriton, Name-based Routing Protocol Specification, in progress, 1999
- [4] Hugh Holbrook and David R. Cheriton, IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications, Proc. ACM SigComm'99 Cambridge, MA Sept. 1999.
- [5] Anonymous (to allow for blind review), Name-enabled Routing and Directory Services in TRIAD, in progress, 2000.
- [6] P. Mockapetris, Domain Names - Concepts and Facilities, RFC 882, November, 1983 (obsoleted by RFC 1034 and 1035).
- [7] D. Eastlake, Domain Name Security Extensions, RFC 2535, March 1999.
- [8] P. Ferguson, H. Berkowitz, Renumbering: What is it and why do I want it anyway, RFC 2071, January 1997.
- [9] S. Deering and R. Hinden, IP Version 6 Addressing Architecture, RFC 2372, July 1998.
- [10] D.R. Cheriton, Local Networking and Internetworking in the V-System, Proc. 8th Data Communication Symposium, IEEE/ACM, 1983
- [11] V. Jacobson, LNat — Large-scale IP via Network Address Translation, working draft, Lawrence Berkeley Labs, Jan. 1992.
- [12] E. Egevang and P. Francis, The IP Network Address Translator (NAT) RFC 1631, May 1994.
- [13] Y. Rekhter, B. Moskowitz, D. Karrenberg and G. de Groot, Address Allocation for Private Internets, RFC 1597, March 1994.
- [14] E. Rosen, A. Viswanathan and R. Callon, Multi-protocol Label Switching Architecture, work-in-progress, Internet draft, 1999.
- [15] M. Borella, D. Grabelsky, J. Lo, Realm Specific IP: Protocol Specification, draft-ietf-nat-rs-ip-protocol-03.txt, Oct. 1999 (work in progress).
- [16] M. Kaat, Overview of IAB 1999 Network Layer Workshop, draft-ietf-iab-ntwlyrws-over-01.txt, work in progress, Oct. 1999
- [17] NAT Working Group, P. Srisuresh and Matt Holdrege, IP Network Address Translator (NAT) Terminology and Considerations draft-ietf-nat-terminology-00.txt, work in progress, July 1998

- [18] S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol (S-BGP)", to appear in IEEE Journal on Selected Areas in Communication, 1999.
- [19] Internet Domain Survey, July 1999, <http://www.isc.org/ds/>.
- [20] Internet Performance Measurement and Analysis project, <http://www.merit.edu/ipma/>.
- [21] D. R. Cheriton and T.P. Mann, Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance. ACM TOCS, May 1989
- [22] Wireless Application Protocol Forum <http://www.wapforum.org>.
- [23] Xerox Network Services Specification, Xerox Corporation, 1980.

Distributed Packet Rewriting and its Application to Scalable Server Architectures*

Azer Bestavros Mark Crovella Jun Liu

Computer Science Department

Boston University

Boston, MA 02215

{best, crovella, junliu}@cs.bu.edu

David Martin†

Dept of Math and CS

University of Denver

Denver, CO 80208

dm@cs.du.edu

Abstract

To construct high performance Web servers, system builders are increasingly turning to distributed designs. An important challenge that arises in such designs is the need to direct incoming connections to individual hosts. Previous methods for connection routing (Layer 4 Switching) have employed a centralized node to handle all incoming requests. In contrast, we propose a distributed approach, called Distributed Packet Rewriting (DPR), in which all hosts of the distributed system participate in connection routing. DPR promises better scalability and fault-tolerance than the current practice of using centralized, special-purpose connection routers. In this paper, we describe our implementation of four variants of DPR and compare their performance. We show that DPR provides performance comparable to centralized alternatives, measured in terms of throughput and delay. Also, we show that DPR enhances the scalability of Web server clusters by eliminating the performance bottleneck exhibited when centralized connection routing techniques are utilized.

1. Introduction

The phenomenal, continual growth of the World Wide Web (Web) is imposing considerable strain on Internet resources, prompting numerous concerns about the Web's continued viability. In that respect, one of the most common bottlenecks is the performance of Web servers—popular ones in particular. To build high performance Web servers, designers are increasingly turning to distributed systems. In such systems, a collection of hosts work together to serve Web requests. Distributed designs have the potential for scalability and cost-effectiveness; however, a number of challenges must be addressed to make a set of hosts function efficiently as a single server.

Connection Routing: Consider the sequence of events that occur as a result of a client requesting a document from a Web server. First, the client resolves the host's domain name to an initial IP address. Second, the IP address itself may represent a distributed system, and one of the hosts in the system must be chosen to serve the request. There are many ways to perform the first mapping (from domain name to initial IP address). For example, this mapping could be coded in the application as is done within Netscape Navigator to access Netscape's Home Page [8]. Alternately, this mapping could be done through DNS by advertising a number of IP addresses for a single domain name. Similarly, there are many ways to perform the second mapping (from initial IP address to actual host). For example, this mapping could be done at the application level, using the HTTP redirection approach [1] or using a dispatcher at the server [2, 17].

While initial attempts to implement connection routing for scalable Web servers focused on using the mapping from domain names to IP addresses [10], recent attempts have focused on the second kind of mapping (IP addresses to hosts) because of the potential for finer control of load distribution. One common feature of all of these attempts (whether proposed or implemented) is that a centralized mechanism is employed to perform the mapping from IP addresses to hosts. Examples include the Berkeley MagicRouter [2], the Cisco Local Director [17], and IBM's TCP Router [7] and Network Dispatcher [9].

Distributed Connection Routing using DPR: In contrast, DPR is a technique that allows the mapping between IP address and host to be implemented in a *distributed*, efficient, and scalable fashion. In particular, DPR can be viewed as a distributed method of mapping m IP addresses to n servers.¹ Using DPR, every host in a Web server cluster acts *both* as a server and as a connection router. Thus, unlike existing solutions that rely on a single, centralized connection router, DPR enables both the service and the routing responsibilities to be

*This work was partially supported by NSF research grants CCR-9706685 and CCR-9501822.

†Research completed while co-author was at Boston University.

¹If $m = 1$, then DPR becomes similar to the centralized solutions mentioned above—the difference being that DPR allows *both* packet routing and service to be combined on the same node.

shared by all hosts in the cluster. Distributing the connection routing functionality allows for true scalability, since adding a new host to the cluster automatically adds enough capacity to boost *both* Web service and connection routing capacities.

To illustrate the benefits of using DPR, consider the problem of scaling up a Web site that initially consists of a single server host. Adding a second server host using typical existing solutions (for example, Cisco's Local Director [17], or IBM's NetDispatcher [9]) requires using special-purpose hardware to distribute incoming HTTP requests between the two server hosts. This kind of centralized solution provides connection routing capacity that far surpasses what a two-host server is likely to require. In other words, the upgrade path (and hence the price tag) for a centralized solution is not truly incremental: the two-host server will be roughly three times the cost (if an ordinary PC is used as a centralized router) and may reach ten times the cost of a single-host server.

An additional, important issue for many content providers is that the centralized solution creates a single-point-of-failure in the system, which leads to even more costly solutions such as using a second, standby connection router. Thus for mission-critical Web sites, centralized connection routing escalates the imbalance in capacity between connection routing and connection service. These problems disappear when using a DPR-based architecture. Adding a second server to the site requires no special hardware, introduces no single-point-of-failure, and utilizes the added capacity (and hence dollars spent) to scale both the connection routing and connection service capacities equally.

Paper Contribution and Scope: The novelty of DPR lies in its *distribution* of the connection routing protocol (Layer 4 Switching), which allows all hosts in the system to participate in request redirection, thereby eliminating the practice of using a special purpose connection router to achieve that functionality.

DPR is one of the salient features of COMMONWEALTH—an architecture and prototype for scalable Web servers being developed at Boston University. The design of DPR is driven by a large set of goals that the COMMONWEALTH architecture strives to achieve. These goals are:

1. **Transparency:** Clients should not be exposed to design internals. For example, a solution that allows a client to distinguish between the various servers in the cluster—and hence target servers individually—is hard to control.
2. **Scalability:** Increasing the size of the cluster should result in a proportional improvement in performance. In particular, no performance bottlenecks should prevent the design from scaling up.
3. **Efficiency:** The capacity of the cluster as a whole should be as close as possible to the total capacity of its constituent servers. Thus, solutions that impose a large overhead are not desired.
4. **Graceful Degradation:** The failure of a system component should result in a proportional degradation in the

offered quality of service. For example, a solution that allows for a single point of failure may result in major disruptions due to the failure of a miniscule fraction of the system.

5. **Connection Assignment Flexibility:** Connection assignment techniques should be flexible enough to support resource management functionalities—such as admission control and load balancing.

In the remainder of this paper we show how DPR supports these goals in the construction of the COMMONWEALTH server. In the next section we review related work and show why DPR is different from previous proposals for connection routing in Web servers. Then in Section 3 we describe the design tradeoffs for DPR and the variants of DPR that we have implemented and tested in our laboratory. In Section 4 we show performance results using DPR, indicating that DPR induces minimal overhead and that it achieves performance scalability superior to that achievable using existing centralized connection routing. Finally, in Section 5 we conclude with a summary.

2. Related Work

Preliminary work on scalability of Web servers has been performed at NCSA [10] and DEC WRL [14]. In both cases, load is balanced across server hosts by providing a mapping from a single host name to multiple IP addresses. In accordance with DNS standard, the different host IP addresses are advertised in turn [16]. In addition to its violation of the transparency property discussed in the previous section, both the NCSA and DEC WRL studies observe that this "Round Robin DNS" (RR-DNS) approach leads to significant imbalance in load distribution among servers. The main reason is that mappings from host names to IP addresses are cached by DNS servers, and therefore can be accessed by many clients while in the cache. The simulations in [7] suggest that, even if this DNS caching anomaly is resolved, the caching of Host-to-IP translations *at the clients* is enough to introduce significant imbalance.

Rather than delegating to DNS the responsibility of distributing requests to individual servers in a cluster, several research groups have suggested the use of a local "router" to perform this function. For example, the NOW project at Berkeley has developed the MagicRouter [2], which is a packet-filter-based approach [13] to distributing network packets in a cluster. The MagicRouter acts as a switchboard that distributes requests for Web service to the individual nodes in the cluster. To do so requires that packets from a client be forwarded (or "rewritten") by the MagicRouter to the individual server chosen to service the client's TCP connection. Also, it requires that packets from the server be "rewritten" by the MagicRouter on their way back to the client. This *packet rewriting* mechanism gives the illusion of a "high-performance" Web Server, which in reality consists of a router and a cluster of servers. The emphasis of

the MagicRouter work is on reducing packet processing time through "Fast Packet Interposing", not on the issue of balancing load. Other solutions based on similar architectures include the Local Director by Cisco [17] and the Interactive Network Dispatcher by IBM [9].

An architecture slightly different from that of the MagicRouter is described in [7], in which a "TCP Router" acts as a front-end that forwards requests for Web service to the individual back-end servers of the cluster. Two features of the TCP Router differentiate it from the MagicRouter solution mentioned above. First, rewriting packets from servers to clients is eliminated. To do so requires modifying the server host kernels, which is not needed under the MagicRouter solution. Second, the TCP Router assigns connections to servers based on the state of these servers. This means that the TCP Router must keep track of connection assignments.

The architecture presented in [11] uses a TCP-based switching mechanism to implement a distributed proxy server. The motivation for this work is to address the performance limitations of *client-side* caching proxies by allowing a number of servers to act as a single proxy for clients of an institutional network. The architecture in [11] uses a *centralized* dispatcher (a Depot) to distribute client requests to one of the servers in the cluster representing the proxy. The function of the Depot is similar to that of the MagicRouter. However, due to the caching functionality of the distributed proxy, additional issues are addressed—mostly related to the maintenance of cache consistency among all servers in the cluster.

3. Implementation of DPR

As described in Section 1, our goals in developing DPR were transparency, scalability, efficiency, fault tolerance, and flexibility in connection assignment. Previous centralized approaches (described in Section 2) have focused on transparency and load balance: these are natural features deriving from a design using centralized routing. The two dominant styles of centralized routing are shown in Figure 1 (a) and (b). Figure 1 (a) shows the MagicRouter style, in which packets traveling in both directions are rewritten by a centralized host. Figure 1 (b) shows the TCP router style, in which only packets traveling from the clients are rewritten, still by a centralized host. An important advantage of the TCP router style is that the majority of bytes in a Web server flow from the server to the client, and these packets do not require rewriting.

In contrast to centralized approaches, we seek to address our wider set of goals, which also include scalability and fault tolerance. As a result we adopt a distributed approach to TCP routing, namely distributed packet rewriting. Under DPR, each host in the system provides both Web service and packet routing functions, as shown in Figure 1 (c). Under DPR the structure of any connection is conceptually a loop passing through three hosts (client and two server hosts). The entire set may have no hosts in common with another connection on

the same distributed server. We refer to the first server host to which a packet arrives as the *rewriter*, and the second host as the *destination*.

Centralized schemes place the rewriting task within the routers connecting a distributed web server to the internet (or as close to such routers as possible). DPR instead transfers this responsibility to the Web servers it concerns. This can be seen as an instantiation of the end-to-end argument: the choice of the final server is essentially a service-specific decision, and so should be made as close as possible to the service points rather than being distributed throughout general-purpose network components.

Another important advantage of DPR is that the amount of routing bandwidth scales with the size of the system, in contrast to the centralized approaches. Furthermore, since the routing function is distributed, this system can not be wholly disabled by the failure of a single node—as is possible under centralized approaches.

The DPR scheme assumes that requests arrive at the individual hosts of the server. This can occur in a number of ways. The simplest approach (which we currently use) is to distribute requests using Round-Robin DNS. Although requests may well arrive in an unbalanced manner because of the limitations of RR-DNS, hosts experience balanced demands for service because of the redistribution of requests performed by DPR.

Design Tradeoffs

Two design issues arise in determining the specific capabilities of a DPR implementation. First, will routing decisions be based on stateless functions, or will it require per-connection state? Second, how should rewritten traffic be carried on the server network? The following sections investigate these questions and describe decisions made in our various implementations extending the Linux 2.0.30 kernel with DPR support.

Stateless vs Stateful Routing:

It is possible to balance load across server hosts using a stateless routing function, *e.g.*, a function that computes a hash value based on the source and destination IP and TCP port addresses of the original packet. On the other hand, more sophisticated load balancing policies may require more information than what is contained in the packets, for example, knowledge of load on other hosts. In this case, each rewriting host must maintain a routing table with an entry for each connection that is currently being handled by that host.

Stateless Approach: In the stateless approach, we use a simple hash function on the client's IP address and port number to determine the destination of each packet. Since the client's IP/port forms a unique key for requests arriving at the server, this function is sufficient to distribute requests.

Using server logs from the BU Web site in simple simulations, we have verified that our hash function is effective

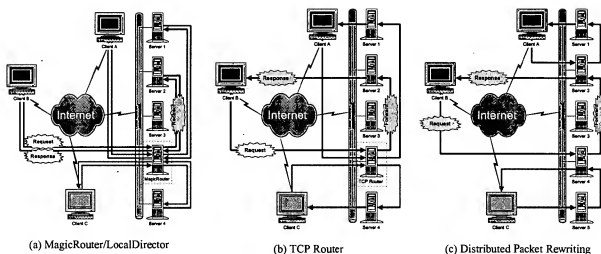


Figure 1. Illustration of various architectures for distributed Web Servers

at balancing load (in terms of hits per server over time) for actual client request arrivals. An important factor in this success is the use of the client port number as an input to the hash function; the client's TCP layer indirectly chooses the final server when it selects its ephemeral TCP port. Successive port numbers from the same client should map to different server hosts, dispersing each burst across servers, and thus alleviating the imbalance due to the burstiness of client requests [7].

Although stateless implementations are lightweight and the resulting server load distributions are acceptable, we must keep in mind the inability of the rewriter to route a connection based on other factors (such as end-server load, distance, availability, or the necessity that successive requests be routed to the same host for correct session semantics).

Stateless/LAN Implementation: This variant takes the simplicity and speed of stateless MAC address rewriting to an extreme. Because no state is stored, the additional code and data required is small. The Stateless/LAN implementation simply overwrites the MAC address of the packet and retransmits it on the LAN. The simplicity of the transformation allows rewriting to occur in the context of the network device driver, namely, in the kernel routine that device drivers use to register incoming packets. This implementation thus receives, rewrites, and retransmits packets all within a single interrupt service cycle; furthermore, no device-specific modifications are required.

While rewriting the entire request in the interrupt routine provides performance virtually indistinguishable from that of a dedicated rewriting router (see Section 4), our Stateless/LAN implementation is not practical. When Stateless/LAN processes fragmented packets, only the first IP fragment contains the necessary TCP port information to ensure proper delivery and subsequent fragments are misrouted.

Because of this shortcoming, we used this implementation only as an indication of an upper bound on the performance that can be achieved with DPR-style techniques.

Stateful Approach: In the stateful approach, the packet routing decision is based on more information than is contained in the packet. For example, a stateful approach is necessary in order to route connections based on the current load on each server host. Most of our efforts have concentrated on this approach.

Stateful Implementation: In the stateful method, rewriters must track TCP connection establishment and termination. A table of connections currently being rewritten is maintained by each host and is consulted in order to rewrite each packet. In implementing these functions we were able to adapt features from code already present in the Linux kernel that supports *IP Masquerading*. *IP Masquerading* [12] was developed to allow multiple hosts to function behind a firewall without valid IP addresses. Thus, *IP Masquerading* supports connections that are initiated from the "hidden" hosts. In order to support a distributed server, we need to support connections connecting to the hidden hosts.

Using the *IP Masquerading* functions adapted to support a distributed server, the rewriter has considerable freedom to choose a destination when it receives the first packet of a client's TCP stream. After noting its decision in a state table, it then forwards each packet associated with the connection using either the MAC rewriting or IPIP encapsulation technique, depending on the network location of the destination.

At present, the routing decision for a newly observed connection is made by simply obtaining the next entry in a ring of server addresses. This ring is extended to user space through a `setsockopt(2)` system call. By populating the ring intelligently, a user daemon can adjust the rewriting policy as server conditions change.

We note that independently and at approximately the same time as our work, Clarke developed a general-purpose TCP forwarding kernel extension based on IP Masquerading [5] which can also be used to support implementation of distributed Web servers.

Addressing Techniques:

There are two approaches to addressing packets bound for another host in a multiple-server environment, depending on whether the original destination IP address must be communicated from the rewriter to the destination host.

The first approach is appropriate when there is only one published IP address for the whole Web server cluster (as would be the case when a centralized connection router is used). In this case, the original packet's destination IP address (IP_1) is replaced with that of the new destination (IP_2) and then the packet is routed normally. When the new destination transmits packets to the client, it must be careful to replace its IP source address (IP_2) with that of the rewriter (IP_1), because the client believes its connection to be with the rewriter (*i.e.* IP_1). Every host in the Web server cluster knows that its outbound traffic should always bear the source address IP_1 , so the address IP_1 need not explicitly appear in rewritten packets.

One consequence of this technique is that the IP and TCP checksums need to be adjusted when rewriting, since they both depend on the destination IP addresses in the packet. (In practical terms, only the IP checksum is important, since IP routers do not examine the payload of IP packets they encounter [3]. However, firewalls and other types of "smart routers" might in fact examine the TCP checksum, so it is advisable to recompute it as well.)

The second approach applies to systems with more than one published IP address, as in DPR. In a DPR system, a mechanism is needed to communicate both the original address (IP_1) and the rewritten address (IP_2) in packets sent between the rewriter and the destination hosts so that the destination knows how to populate the IP source address field. The most efficient method we used was to rewrite the MAC address of the packet and retransmit, leaving the original packet's IP addresses and checksums undisturbed. (This is how Internet hosts normally send packets through a gateway.) Although fast, the method only works if both servers are located on the same LAN. If the servers are on different LANs, then IP-level routing is necessary. In this case we tunnel the original packet to IP_2 with IP/IP encapsulation as described in RFC2003[15]. When the packet arrives at IP_2 , the outer IP/IP header is discarded and the inner header is interpreted.

Whether MAC rewriting or IP/IP encapsulation is used, the server with primary address IP_2 eventually receives and processes an IP packet bearing the original destination address IP_1 . Therefore, each server must be configured to respond to all of the possible original destination addresses (such as IP_1) in addition to its own primary address. In our Linux implementation, this was just a matter of adding loop-

back alias devices with the extra addresses.

4. Performance Evaluation

In this section we describe the performance of DPR variants. We have two goals: first, to characterize the overheads present in DPR; and second, to study the scalability of DPR as compared to centralized connection routing.

To address these two goals we ran two series of experiments. The first series used a small network in determining the performance overhead of Stateful DPR and Stateless/LAN DPR when compared to a centralized connection router, and to baseline cases involving no connection routing. For this set of experiments we used the SPECweb96 [6] benchmarking tool because it places relatively smooth loading on the server over time.

The second series of experiments concentrated on exploring the scalability of Stateful DPR compared to centralized connection routing. Since our goal in this section was to explore how DPR would behave under realistic conditions, we used the Surge reference generator [4] to provide the server workload. Surge is a tool developed as part of the COMMONWEALTH project that attempts to accurately mimic a fixed population of users accessing a Web server. It adheres to six empirically measured statistical properties of typical client requests, including request size distribution and inter-arrival time distribution. As a result, it places a much busier load on the server than does SPECweb96. In addition, while SPECweb96 uses an open system model (requested workload is specified in GETs/sec), Surge adopts a closed system model (workload is generated by a fixed population of users, which alternate between making requests and lying idle). As a result, Surge's workload intensity is measured in units of User Equivalents (UEs).

In both series of experiments we restricted our configurations to a single LAN in order to provide repeatable results. Although the LAN was not completely isolated during our measurements, only a negligible amount of unrelated traffic (mostly ARP requests) was present.

4.1. Performance Overhead of DPR

As described above, SPECweb96's principal independent parameter is the requested throughput, measured in HTTP GETs per second. The measured results of each experiment are the achieved throughput (which may be lower than what was requested) and the average time to complete an HTTP GET (measured in msec/GET). For each experiment, we ran SPECweb96 for the recommended 5 minute warmup, after which measurements were taken for 10 minutes. System hosts (both clients and servers) consisted of Hewlett-Packard Vectra PCs, each having a 200MHz Pentium Pro processor, 32 MB of memory, and a SCSI hard drive. Servers ran Linux 2.0.30 on Linux ext2 filesystems, while clients ran Windows NT 4.0. We used the NT Performance Monitor to ensure that

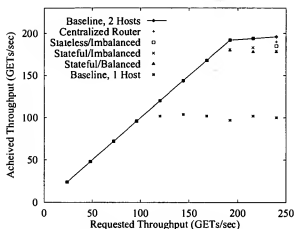


Figure 2. Throughput of DPR Variants

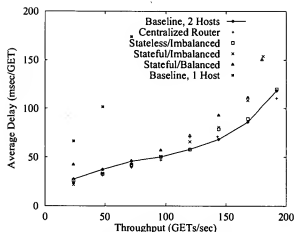


Figure 3. Request Delay of DPR Variants

our clients had capacity to spare when our servers became saturated. The LAN was a 100 Mbit/sec Hewlett-Packard AnyLAN switch; this star network is frame-compatible with Ethernet, but it also uses a round-robin schedule together with a client sensing mechanism so that packet collisions do not occur. The Web servers used were Apache 1.2.4.

We describe the results of six experiments:

Baseline 1-Host. This experiment tests the performance of a single, unmodified server driven by a single client.

Baseline 2-Host. This experiment consists of two simultaneous copies of the Baseline 1-Host experiment. It uses two clients and two servers, and each client sends requests to only one server.

Centralized Router. This experiment consists of two clients sending requests to a centralized connection router, which then distributes the load evenly between two servers. The Centralized Router implementation is our Stateful DPR configured to redirect all connections to the other two servers (i.e. the routing function does not compete with local web service).

Stateless/Imbalanced. This experiment uses the Stateless/LAN variant of DPR, running on two hosts. Two clients generate requests, but they send *all* requests to one of the server hosts, which then redistributes half of them.

Stateful/Imbalanced. This experiment uses the Stateful variant running on two hosts. Again two clients generate requests, sending all requests to one host, which redistributes half of them.

Stateful/Balanced. This experiment again uses the Stateful variant, but now the two clients generate equal amounts of requests for each server host. Each host then redistributes half of its requests, sending them to the other server.

Baseline 1-Host and Baseline 2-Host define the range of possible performance for the systems under study, with Baseline 2-Host defining the best performance that might be expected

from a 2-host server system. The Centralized Router results represent the performance of the most common alternative to DPR, and show the effect of removing the packet rewriting function from the server hosts. Note that each packet travels through two server nodes in the DPR and Centralized Router cases, and through only one server node in the Baseline cases.

The Stateless/Imbalanced and Stateful/Imbalanced experiments serve to show the worst possible performance of DPR, i.e., when the arriving request load is maximally imbalanced (all requests to one host). The Stateful/Balanced experiment allows comparison of the best and worst possible load arrival distributions for DPR.

Throughput:

In Figure 2 we show the achieved throughput of each experimental system as a function of the requested throughput. The Baseline 1-Host case saturates at about 100 GETs/sec, and the Baseline 2-Host case at the corresponding level of about 200 GETs/sec. In between the experiments fall into two groups: the Stateful experiments saturate at about 180 GETs/sec, while the Stateless/Imbalanced and Centralized Router saturate at about 195 GETs/sec. The fact that the Stateful/Balanced and Stateful/Imbalanced show nearly identical performance indicates that when requests arrive in a highly imbalanced way and all packet rewriting occurs on only one host, DPR is still able to achieve good throughput. This comparison indicates that the performance demand of packet rewriting is quite moderate, and so adding a packet rewriting function to a host already performing Web service does not represent a significant additional burden.

Comparing the Stateful and Stateless cases, we see that the Stateless case performs indistinguishably from the Centralized Router case, and they both are equivalent to the Baseline 2-Host case (in which no packet rewriting is taking place at all). The similarity of the Stateless to the Baseline 2-Host case shows that the performance cost of packet rewriting in

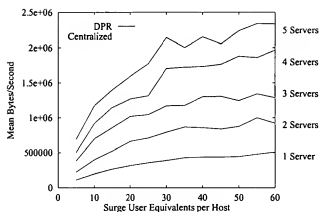


Figure 4. Throughput Comparison

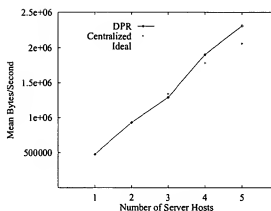


Figure 5. Scalability Comparison

the Stateless/LAN implementation is negligible.

An important implication of the similarity of the Stateless and Centralized Router cases is that the centralized connection routing architecture is not cost-effective. This is because an entire node has been allocated to the connection routing task (there are three nodes in the Centralized Router case and only two in the DPR cases). Thus the additional cost of adding a specialized connection router to a small system may not be justified. It is just as efficient, and cheaper, to use the server hosts already present to perform the connection routing function. This point will be reinforced by our results in Section 4.2 on the scalability of the DPR architecture compared to the centralized routing architecture.

Delay:

In addition to providing high throughput, it is important to verify that DPR does not add unacceptable delays to the system. In Figure 3 we show the average response time of an HTTP GET (in msec/GET) as a function of system throughput, for the same six experiments. In this figure we only plot those points for which achieved and requested throughput are nearly equal, so throughput does not reach quite the same maximum values as in Figure 2. Figure 3 shows that the experiments break into the same groupings as before. Again, the Stateful/Balanced and Stateful/Imbalanced cases show approximately similar performance. Furthermore the Stateless case shows approximately similar delays to the TCP Router and the Baseline 2-Host cases.

Since packets travel through an additional server node in the DPR and TCP Router cases as compared to the Baseline 2-Host case, there is a potential for greater delay in those cases. However, it appears that the additional delays induced by the additional hop are small compared to the average response time for an HTTP GET. The response time of an average HTTP GET under SPECweb96 is in the range of 25 to 150 milliseconds on a LAN. Were the system serving packets over the global Internet, response times would be even greater since the added round-trip times would be tens to hundreds of

milliseconds as well. The addition of additional packet processing due to Stateless/LAN DPR, which appears to be on the order of tens to hundreds of microseconds, is a negligible additional cost for a Web server application.

4.2. Scalability of DPR

The previous section showed that the overheads of DPR were no greater than that of a centralized connection router, and that even when connection routing load was completely unbalanced, system performance did not suffer. These results suggest that DPR should show good scalability, but it is still necessary to evaluate DPR's scalability in practice. For comparison purposes we also evaluate the centralized connection routing case.

The scalability series of experiments took place on different equipment than the performance overhead experiments. All of the Web/DPR servers were Dell Dimension PCs with 64 MB of memory and IDE hard drives running Linux 2.0.30 and Apache 1.2.1. Four of the Web servers had 200 MHz Pentium Pro processors, and one had a 233 MHz Pentium II. The latter system appears in our results as the fourth Web server in both DPR and centralized connection router experiments. Our fastest system, a 266 MHz Pentium II, was used only as a connection router. Both clients and servers used Linux ext2 file systems. The LAN was a 12-port 3Com SuperStack II Switch 3000 10/100 running Ethernet in full duplex at 100Mb/s. The total bandwidth measured during the experiments show that network capacity was not a limiting factor; we also observed that our clients were able to saturate our servers before reaching their own capacity.

As described above, for these experiments we used the Surge load generator. In the DPR cases, we configured Surge so that equal amounts of traffic were directed at each server host. In an N host system, each DPR host serves $1/N$ of the requests locally and distributes $(N-1)/N$ of the requests equally to the other hosts in the system. In adopting this routing policy, our results for DPR are quite conservative.

A better policy that is still quite practical would be for each server host to only redirect requests that arrive when the host is loaded above the system average; in that case, a fraction of requests much smaller than $(N - 1)/N$ would be redirected, and the overall performance of the DPR system would be better than that reported here.

In our experiments we compare N -host DPR systems to centralized routing configurations consisting of N server hosts *plus* a connection router. By doing so, we emphasize the scalability difference between the architectures. However these tests do not compare equivalent systems in terms of hardware costs; as described above, it is more cost-effective to organize a system of N hosts in a DPR architecture than to set aside one host solely for connection routing.

In order to scale the demand placed on the server systems as the number of server hosts grows, it is necessary to proportionally increase the number of User Equivalents used in Surge. For this reason we report results in terms of User Equivalents per server host.

The achieved throughput for a range of both DPR systems and centralized routing systems is shown in Figure 4. This figure shows that for small systems (2-3 hosts), DPR and centralized routing behave approximately equivalently. However for larger systems (4-5 hosts), the centralized approach seems to show lower maximum throughput than the DPR approach. This evidence that the centralized node is beginning to become a bottleneck is supported by the fact that the difference between the two systems becomes more pronounced as the demand grows.

To illustrate the onset of a bottleneck effect in the centralized routing case, we show the peak throughput achieved as a function of the size of the system in Figure 5. Peak throughput was obtained in each case by averaging the throughput over the range 50-60 UEs per host (which in each case was where system saturation was judged to have set in). The figure also shows the "ideal" throughput obtained by simply scaling up the throughput obtained by a single unmodified host.

This figure shows that DPR obtains near-perfect speedup for server systems up to five hosts in size. In contrast, the centralized routing architecture seems to show signs of inefficiency at larger sizes; on four hosts, maximum throughput under centralized routing has dropped to 94% of ideal, and on five hosts the centralized system is only 86% efficient.

5. Summary

In this paper we have proposed and experimentally evaluated a protocol for routing connections in a distributed server without employing any centralized resource. Instead of using a distinguished node to route connections to their destinations, as in previous systems, Distributed Packet Rewriting (DPR) involves *all* the hosts of the distributed system in connection routing. The benefits that DPR presents over cen-

tralized approaches are considerable: the amount of routing power in the system scales with the number of nodes, and the system is not completely disabled by the failure of any one node. DPR allows more cost-effective scaling of distributed servers, and as a result more directly supports the goals of the COMMONWEALTH project.

References

- [1] D. Anderson, T. Yang, V. Holmedahl, and O. Ibarra. SWEB: Towards a Scalable World Wide Server on Multicomputers. In *Proceedings of IPPS'96*, April 1996.
- [2] E. Anderson, D. Patterson, and E. Brewer. The MagicRouter: An application of fast packet interposing. <http://HTTP.CS.Berkeley.EDU/~eanders/projects/magicrouter/osd196-mr-submission.ps>, May 1996.
- [3] F. Baker. IETF RFC1812: Requirements for IP Version 4 Routers. See <http://ds.internic.net/rfc/rfc1812.txt>.
- [4] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151-160, Madison, WI, June 1998.
- [5] S. Clarke. Port Forwarding in Linux. See description at <http://www.ox.compsoc.org.uk/~steve/portforwarding.html>.
- [6] T. S. P. E. Corporation. Specweb96. <http://www.specbench.org/org/web96/>.
- [7] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of IEEE COMPCON'96*, pages 85-92, 1996.
- [8] S. Garfinkel. The Wizard of Netscape. *WebServer Magazine*, pages 58-64, July/August 1996.
- [9] IBM Corporation. The IBM Internet Network Dispatcher. See <http://www.ics.raleigh.ibm.com/netdispatch>.
- [10] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. In *Proceedings of the First International World-Wide Web Conference*, May 1994.
- [11] K. Law, B. Nandy, and A. Chapman. A Scalable and Distributed WWW Proxy System. Technical report, Nortel Limited Research Report, 1997.
- [12] Linux IP Masquerade Resource. See <http://ipmasq.home.ml.org>.
- [13] J. Mogul, R. Rashid, and M. Accetta. The Packet Filter: An Efficient Mechanism for User-level Network Code. In *Proceedings of SOSP'87: The 11th ACM Symposium on Operating Systems Principles*, 1987.
- [14] J. C. Mogul. Network behavior of a busy Web server and its clients. Research Report 95/5, DEC Western Research Laboratory, Oct. 1995.
- [15] C. Perkins. IETF RFC2003: IP Encapsulation within IP. See <http://ds.internic.net/rfc/rfc2003.txt>.
- [16] R. J. Schemers. lbnamed: A Load Balancing Name Server in Perl. In *Proceedings of LISA'95: The 9th Systems Administration Conference*, 1995.
- [17] C. Systems. Scaling the Internet Web Servers: A white Paper. <http://www.cisco.com/warp/public/751/lodir/scale.wp.htm>, 1997.

Memex: A browsing assistant for collaborative archiving and mining of surf trails

Soumen Chakrabarti

Sandeep Srivastava

Mallela Subramanyam

Mitul Tiwari

Indian Institute of Technology Bombay

soumen,sandy,manyam,mits@cse.iitb.ernet.in

Abstract

Keyword indices, topic directories, and link-based rankings are used to search and structure the rapidly growing Web today. Surprisingly little use is made of years of browsing experience of millions of people. Indeed, this information is routinely discarded by browsers. Even deliberate bookmarks are stored in a passive and isolated manner. All this goes against Vannevar Bush's dream of the *Memex*: an enhanced supplement to personal and community memory.

We propose to demonstrate the beginnings of a 'Memex' for the Web: a browsing assistant for individuals and groups with focused interests. Memex blurs the artificial distinction between browsing history and deliberate bookmarks. The resulting glut of data is analyzed in a number of ways at the individual and community levels. Memex constructs a topic directory customized to the community, mapping their interests naturally to nodes in this directory. This lets the user recall topic-based browsing contexts by asking questions like "What trails was I following when I was last surfing about *classical music*?" and "What are some popular pages in or near my community's recent trail graph related to *music*?"

1 Motivation

Three paradigms have emerged for exploring the Web: keyword search, directory browsing, and following links. Popular search engine and directory sites are visited tens of millions of times per day. We speculate that the total number of clicks per day is orders of magnitude larger. This third source of information, the browsing history of millions of Web users over several years, an information source that dwarfs the scale of the Web itself, is almost entirely discarded by browsers as 'history'. Deliberate 'bookmarks' are preserved, but passively, in browser-dependent formats; this separates them from the dominant world of HTML hypermedia, even if their owners were willing to share them (as they are, in our experience, with all but a small section of their browsing activity).

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

In 1945, Vannevar Bush dreamt of *Memex*: an enhanced, intimate supplement to personal and community memory [2]. Assisted by a Memex for the Web, a surfer can ask:

- What was the URL I visited about six months back regarding compiler optimization at Rice University?
- What was the Web neighborhood I was surfing the last time I was looking for resources on classical music?
- Are there any popular sites, related to my (Web) experience on classical music, that have appeared in the last six months?
- How is my ISP bill divided into access for work, travel, news, hobby and entertainment?
- What are the major topics relevant to my workplace? Where and how do I fit into that map? How does my bookmark folder structure map on to my organization?
- In a hierarchy of organizations (by region, say) who are the people who share my interest in recreational cycling most closely and are not likely to be computer professionals?

Since Bush proposed Memex, the theme of a 'living' hypermedia into which we "weave ourselves" has been emphasized often, e.g., by Douglas Engelbart¹ and Ted Nelson², and of late by Tim Berners-Lee³ and Jim Gray⁴. Indeed, the current cost/volume ratio of storage makes it unnecessary to delete *anything* from one's Web surfing experience, provided we can make fruitful use of it.

We propose an architecture of a 'Memex' for the Web which can answer the above questions. Memex is a large project involving hypertext data mining, browser plug-in and applet design, servlets and associated distributed database architecture, and user interfaces. We have validated the design using a prototype implementation that we describe here. Memex is currently implemented on Netscape 4.5+. We are currently testing Memex with the help of local volunteers. The Memex service will be made

¹<http://jefferson.village.virginia.edu/elab/hf10035.html>

²<http://www.sfc.keio.ac.jp/~ted/>

³<http://www.w3.org/1999/04/13-tbl.html>

⁴http://research.microsoft.com/gray/papers/MS_TR_99_50_TuringTalk.pdf

publicly accessible⁵. Further details about Memex have been reported elsewhere [4].

2 Client architecture overview

Memex should run on popular browsers. It should be possible to distribute updates and new features effortlessly to users. Hence the Memex client has been designed as an applet. In view of secure firewalls, proxies, and ISPs' restrictions on browser setups, the client should communicate with the server over HTTP. The data transferred should be encrypted, if desired, to preserve privacy.

The user can log on to a Memex server at the level of a department, organization, interest group, ISP, nation or the world. The architecture makes no assumptions about the logical community level at which Memex might be deployed. At any time, the user can choose not to archive surfing actions, archive for private use, or archive for use by the community (Figure 1). If permitted, Memex taps the browser to get the current location and passes this on to the server, which then processes it in many ways.

Apart from a standard full-text search over all pages visited, the Memex client has several function tabs to assist topic-based mining. The editable **folder tab** (Figure 1) provides topic management: this is the means by which users exemplify their interests. Existing bookmarks from Netscape or Explorer can be imported into Memex's editable tree-structured topic view; conversely Memex can export back to these browsers. Apart from implicit history logging, bookmarks can be added to folders while surfing. A user will typically assign a bookmark explicitly to a topic. These assignments are analyzed by the server, which then classifies all surfed pages automatically into these folders. The folder tab can also be used to reinforce or correct the classifier. Memex also uses unsupervised clustering to propose a topic hierarchy [6] over a set of links that the user may want to reorganize. Periodically, the server consolidates all users' public folders and browse history into a topic directory tailored to the needs of that specific community (see §4 and Figure 4).

Users surf on many topics with diverse priorities. Because browsers have only a transient context (one-dimensional history list), surfers frequently lose context when browsing about a topic after a time lapse. Studies have shown that visiting Web pages is best expressed using spatial metaphors: your context is "where you are" and "where you are able to go" next [9]. Memex's topic classifier also helps render the topic-focused **trail tab** (Figure 2). In the trail tab, the left panel shows the user's topic folders. Selecting a folder replays the hypertext graph of recent pages publicly surfed by the community which are

⁵<http://www.cse.illb.ernet.in/~soumen/memex/>

most likely to belong to the selected topic, and thus recreates the user's browsing context.

3 Server architecture overview

On the server side, the system should be robust and scalable. It is important that the server recovers from network and programming errors quickly, even if it has to discard a few client events. The server consists of servlets that perform various archiving and mining functions as triggered by client action, or continually as demons. We prefer servlets to CGI scripts because the client-server interactions exchange complex objects and sometimes have state. We prefer HTTP tunneling also because direct JDBC connections may be refused by many firewalls.

Server state is managed by two storage mechanisms: a relational database (RDBMS) such as Oracle or DB2 for managing metadata about pages, links, users, and topics, and a lightweight **Berkeley DB**⁶ storage manager to support fine-grained term-level data analysis for clustering, classification, and text search. Storing term-level statistics in an RDBMS would have overwhelming space and time overheads.

An interesting aspect of the Memex architecture is the division of labor between the RDBMS and the lightweight storage manager. Planning the architecture was made non-trivial by the need for asynchronous action from diverse modules. There are some user interface-related events that must be guaranteed immediate processing. Typically these are generated by a user visiting a page, or deliberately updating the folder structure. With many users concurrently using Memex, the server cannot analyze all visited pages, or update mined results, in real time. Background demons continually fetch pages, index them, and analyze them w.r.t. topics and folders. The data accesses made by these demons have to be carefully coordinated. This would not be a problem with the RDBMS alone, but maintaining some form of coherence between the metadata in the RDBMS and several text-related indices in Berkeley DB required us to implement a loosely-consistent versioning system on top of the RDBMS, with a single producer (crawler) and several consumers (indexer and statistical analyzers). Figure 3 shows a block diagram of the system.

4 Mining algorithms overview

The stream of data from surfers has to be analyzed in various ways. Some parts of the processing, such as keyword indexing, are mundane. Other parts constitute new algorithms or novel implementations.

For clustering we started with a bottom-up hierarchical agglomerative approach [6]. For classification we started with a Bayesian classifier [3]. Although these simple text-based techniques work reasonably well for

⁶<http://www.sleepycat.com>

average Web pages, bookmarked URLs offer special challenges: people tend to bookmark many "front pages" with less text and more graphics compared to typical Web documents. Surfers may also place two URLs in the same folder for functional reasons, even if the corresponding documents are syntactically dissimilar.

We have implemented two new learning algorithms for Memex. For classification we use a new technique that combines features from text, hyperlink and folder placement to offer significantly boosted accuracy, increasing from a mere 40% accuracy for text-only learners to about 80% with our more elaborate model.

We generalize clustering to finding a new notion of **themes** among the bookmarks. In principle, each user need not design his/her own topic hierarchy, given there are 'standard' ones like *Yahoo!*⁷ and the *Open Directory*.⁸ In practice, these 'universal' hierarchies are neither necessary nor sufficient for individual surfers and focused communities, they are too specialized in most topics, and not sufficiently specialized in the areas in which the community is deeply interested. We propose a new formulation for discovering a topic hierarchy specifically expressing and addressing the interests of the community, refining topics where needed and coarsening where possible. Details of the new classification and theme discovery algorithms are reported elsewhere [4] (also see Figure 4).

Once topic hierarchies for the user community are determined, automatic resource discovery is undertaken by demons to update users about recent and/or authoritative sources, organized by topic [5]. 'Normalizing' all members of the community to themes also lets us represent surfers' interests in a *canonical form*: roughly speaking, a user profile is a set of weights associated with each node of a theme hierarchy; this gives us a means of comparing profiles that is far superior to overlap in sets of URLs. We intend to use this for better collaborative recommendation [10].

5 Related work

Our work is closest in spirit to two well-known systems, *PowerBookmarks*⁹ and the *Bookmark Organizer* [8].

PowerBookmarks is a semi-structured database application for archiving and searching bookmark files via explicit CGI programs. *PowerBookmarks* uses *Yahoo!* for classifying the bookmarks of all users. In contrast, *Memex* preserves each user's view of their topic space, and reconciles these diverse views at the community level. Furthermore, *PowerBookmarks* does not use hyperlink information for classification or for synthesizing themes. The *Bookmark Organizer* is a client-side solution for personal organization, but does

not provide community-level themes or topical surfing contexts. *Purple Yogi*¹⁰ is a client-side software which logs pages visited and clusters them into folders. Then it tunes in on the *Purple Yogi* server to collect additional related material. No community-level mining is involved; *Purple Yogi* explicitly guarantees that user-specific data is stored locally on the user's desktop and never shipped out. Thus scope for valuable collaboration is lost and surfing history becomes inaccessible from other places from which the user might browse.

Other Internet start-ups have been quick to discover the annoyance of surfers maintaining multiple bookmark files and the opportunity of a central, networked bookmark server. We can list several sites which, using Javascript or a plugin, import existing Netscape or Explorer bookmarks and thereafter lets the surfer visit their Web site and maintain it using CGI and Javascript: *Yahoo Companion*¹¹, *YaBoo*¹², *Baboo*¹³, *Bookmark Tracker*¹⁴, and *Backflip*¹⁵ are some examples. Some services like *Third Voice*¹⁶ enables surfers to attach public or private annotations to any page they visit. These are essentially glorified FTP services with none of our extensive server-side analysis.

Several visualization tools have been designed recently that explore a limited radius neighborhood and draw clickable graphs. These are often used for site maintenance and elimination of dead links. Mapuccino and Fetuccino from IBM Haifa are well known examples [7, 1]. Our context viewer could benefit from better hypertext rendering techniques.

References

- [1] I. Ben-Shaul, M. Herscovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalheim, V. Soroka, and S. Ur. Adding support for dynamic and focused search with Fetuccino. In *8th World Wide Web Conference*. Toronto, May 1999.
- [2] V. Bush. As we may think. *The Atlantic Monthly*, July 1945. Online at <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>.
- [3] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organising large text databases into hierarchical topic taxonomies. *Vldb Journal*, Aug. 1998. Invited paper, online at http://www.cs.berkeley.edu/~soumen/VLDB54_3.PDF.
- [4] S. Chakrabarti, S. Srivastava, M. Subramanyam, and M. Tiwari. Archiving and mining community web browsing experience using Memex. In *9th International World Wide Web Conference*, Amsterdam, May 2000.
- [5] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31:1623-1640, 1999. First appeared in the *8th International World Wide Web Conference*¹⁷.

¹⁰<http://www.purpleyogi.com>

¹¹<http://www.yahoo.com/x/cm>

¹²<http://www.yaboo.dk>

¹³<http://www.baboo.com>

¹⁴<http://www.bookmarktracker.com>

¹⁵<http://www.backflip.com>

¹⁶<http://www.thirdvoice.com>

¹⁷<http://www8.org>

⁷<http://www.yahoo.com>

⁸<http://dmoz.org>

⁹<http://www.ccr1.nclab.com/webdb/>

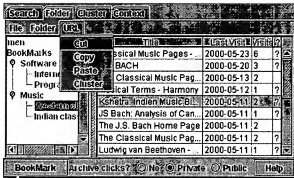


Figure 1: Each user has a personal folder/topic space, which is usually initialized by importing existing browser-specific bookmark folders. The classification demon then classifies all subsequent history elements, marking its guesses by '?'. The user can correct or reinforce the classifier using cut/paste, thus continually improving Memex's models for the user's topics of interest.

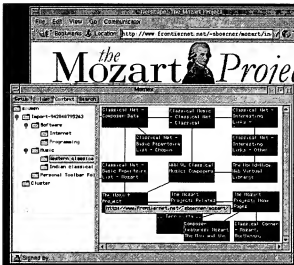


Figure 2: The trail tab shows a read-only view of the user's current folder structure. When the user selects a folder, Memex replays recently browsed pages which belong to the selected (or contained) topic(s), reminding the user of the latest topical context. In the screen-shot above, the chosen folder is /Music/Western Classical. The user can now resume browsing and the display is updated with additional resources related to the topic.

Toronto, May 1999. Available online at <http://www8.org/v8-papers/5a-search-query/crawling/index.html>.

- [6] D. R. Cutting, D. R. Karger, and J. O. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *Annual International Conference on Research and Development in Information Retrieval*, 1993.
- [7] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalheim, and S. Ur. The Shark-Search algorithm-an application: Tailored web site mapping. In *7th World-Wide Web Conference*, Brisbane, Australia, Apr. 1998. Online at <http://www7.ecu.edu.au/programme/fullpapers/1849/cont1849.htm>.

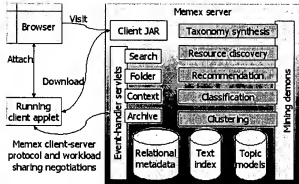


Figure 3: Block diagram of the Memex system, showing the client-server interface, the UI event handlers, the mining demons, and the loosely synchronized data repositories.

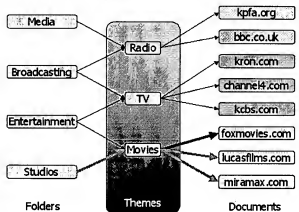


Figure 4: Memex computes, from the document-folder associations of multiple users, a topic taxonomy specifically tailored for the interests of that user population. The taxonomy consists of *themes* which capture common factors in people's interests when they can, while maintaining individuality when they must. Once computed, they can be used to guide resource discovery and collaborative recommendation.

- [8] Y. S. Maarek and I. Z. Ben Shaul. Automatically organizing bookmarks per content. In *Fifth International World-Wide Web Conference*, Paris, May 1996.
- [9] P. P. Maglio and T. Matlock. Metaphors we surf the Web by. In *Workshop on Personalized and Social Navigation in Information Space*, Stockholm, Sweden, 1998.
- [10] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems*, 1998. Online at <http://www.cis.upenn.edu/~ungar/papers/clust.ps>.

APNNed goes Internet

P. Kemper and C. Tepper *

Dept. of Computer Science IV, University of Dortmund
D-44221 Dortmund, Germany
e-mail: {kemper, tepper}@ls4.cs.uni-dortmund.de

Abstract

APNNed is an editor for hierarchical, colored, and stochastic Petri nets. It is an integral part of the APNN toolbox and serves as a graphical user interface for other members of this toolset. Since it is implemented in Java, it is rather platform independent. In this paper we describe recent enhancements to support modeling and analysis of Petri nets with APNNed via internet. We give a brief sketch of technical issues like HTTP-tunneling that need to be solved to have an APNNed applet running at the client side and an APNNed servlet at the server side, that provides load, save, and analysis functionality for nets.

1 Introduction

A major attraction of web applications is its ease of use; a user need not install the specific application but can rely on a general purpose web browser possibly enriched by a set of plug-ins. Furthermore, web applications can be made accessible to an arbitrary or to a restricted set of users with limited effort. A Petri net tool can profit from these advantages in many ways, for example, for teaching modeling with Petri nets in classes, students can work with Petri net tools at their home PC, or for large research projects, in which different groups of researchers interact and need to use and provide tools or applications for each other. However, few Petri net tools make use of this opportunity yet.

In this paper, we briefly describe a recent attempt to enhance the java-implemented Petri net editor APNNed [7] of the APNN toolbox with functionality to perform as a web application. We consider a scenario, in which APNNed applet is run by a web browser. The applet allows to load a set of available models from an HTTP server or to create new Petri net models. Any model can be edited and modified as in the stand-alone APNNed. Fig. 1 shows a screenshot of APNNed as an applet of an HTML page viewed by a browser. Note that, once started, applets are not restricted to the canvas space taken by the web browser but can expand over the whole screen. The applet provides some functionality like the token game by itself and more sophisticated analysis by help of a servlet that resides at the HTTP server and that uses the APNN toolbox. The servlet provides a load and save operation for persistent storage of nets. Obviously, the possibility to save models at the server's site makes sense only in a restricted group of users. In the following, we will not focus on security but on some technical issues to set up the necessary communication among applet and servlet to transfer information on nets, requests for analysis and analysis results. During its implementation, we had to recognize that

*This research is supported by DFG, collaborative research center 559 'Modeling of Large Logistic Networks'.

this case is not standard and among the many Java books that discuss servlets we found only in [10] a substantial treatment of HTTP-tunneling. Since many Petri net tool developers may consider to turn their tools into web applications, we believe that our experiences are useful and such technical issues are of broader interest.

Before we go into these details, let us briefly recall the functionality and architecture of the APNN toolbox. The APNN Toolbox [2] is a collection of analysis tools, all of which are combined by APNNed [7], which serves as a graphical user interface to edit Petri net models and also controls application of analysis tools for a given net, such that results of analysis are subsequently visualized. All tools interact via a textual interface called Abstract Petri net notation (APNN) [3]. The APNN toolbox contains programs for the functional and quantitative analysis. Functional analysis includes the calculation of invariants, based on the algorithm of Martinez and Silva [13], a randomized token game to detect deadlocks, a analysis techniques based on the reachability graph like deciding liveness and modelchecking. Model checkers exists for a computational tree logic (CTL) [12] and a linear temporal logic (LTL). Quantitative analysis is supported by a discrete event simulator and numerical analysis methods based on an associated continuous time Markov chain (CTMC). The APNN toolbox has a clear focus on state based analysis, in which extremely large but finite state transitions systems are represented by hierarchical or modular Kronecker representations [4, 5, 6, 11]. State-based analysis tools interact via a second, a state level interface that describes the reachability set as a multi-way decision diagram and the reachability graph as a set of matrices that are composed by Kronecker operations.

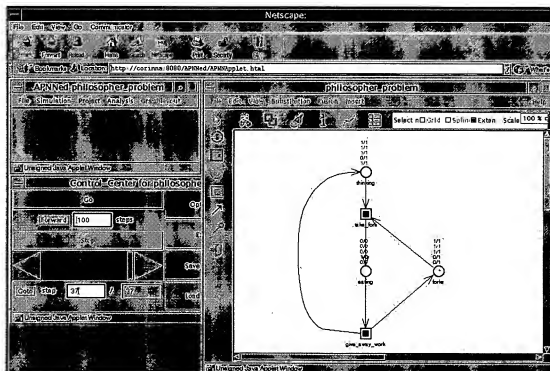


Figure 1: Screenshot of the internet version of APNNed

The paper is structured as follows. Section 2 describes design considerations and identifies four basic operations to achieve an internet version of APNNed. Section 3 describes how to set up an applet-servlet communication (HTTP-tunneling) for an applet, while Section 4 gives the counterpart for the servlet. We summarize in Section 5.

2 Design issues

Web applications are often implemented by Common Gateway Interface (CGI) programs such that a HTTP request of a client results in the instantiation of a process at the server's site to generate a response which often is a web page in HTML. Starting a new process for each HTTP-request causes a significant overhead if the response computation itself only takes a minor effort. A servlet is a similar concept in Java but it runs on threads rather than processes with obvious advantages for performance and the ability to exchange data among threads, such that a servlet can store data over a sequence of requests from a client. Java servlet technology [10] provides web developers with a simple mechanism for extending the functionality of a web server. Servlets are server- and platform-independent and have access to the entire Java APIs [8], [9]. A servlet profits from Java's general properties like portability, performance, re-usability and crash protection (to the extend an existing virtual machine and language implementation can keep pace with expectations).

APNNed is implemented in Java, so that it is a natural choice to use servlets. As mentioned above, our scenario focuses on four simple operations, namely to load, save and analyze nets at the server's site and an additional operation to give information on which nets are available at the server. We do not consider additional topics imposed by storing nets by the server, which nevertheless need to be considered in practice, these include security, restricted access, privacy and sharing of nets, but also traversal of directory structures, disc usage and cpu usage by external clients for analysis etc.

We focus on the applet-servlet communication. The four operations require transfer of an operation identifier plus operation specific parameters as given below:

Operation	Parameter1	Parameter2	Parameter3
load	operation=load	netname=Example	
save	operation=save	net=<APNN description>	netname=Example
analyze	operation = analyze	netname=Example	method=name+parameters
directory	operation=dir		

The information for each operation need to be encoded at the sender and decoded by the receiver. Parameters are rather straightforward, i.e. netname gives the filename of the net, net gives a textual description of the net in APNN format, method gives an identifier for the selected analysis method plus some additional method-specific parameters.

Since applet and servlet are both Java applications, one can transfer serialized Java objects, such that a receiver can pick the complete information for an object just by a single call to *readObject* and one need not programme the parsing and assignment procedure to instantiate an appropriate object. Alternatively, applet and servlet can communicate via buffered input stream to transfer binary or ASCII data. In both cases, an Java *URLConnection* is needed to establish an input stream. The internet version of APNNed uses an object input stream to send and receive objects of class *String*. Clearly, we would use an object stream to exchange net objects but since some internal APNNed classes to represent a net are not serializable yet, we

need to transfer an APNN description as a string object and the receiver needs to parse this string. The encoding of an operation into a string uses an URLEncoder object, e.g., for the load operation it is: `String data = "operation=" + URLEncoder.encode("load") + "netname=" + URLEncoder.encode("<name of Petri net>");` Encoding for other operations is done accordingly.

In the following, we present the necessary steps to call the operations above by an applet and to transfer corresponding answer by a servlet.

3 Starting communication by a client

The applet sends first data to the servlet before the servlet sends back any kind of response. To start the communication, an applet mainly needs to open a connection with an `URLConnection` object and to generate a request with an appropriate output stream. The steps given below give the details and clearly identify the Java classes that are involved, for more information see [10]. The steps need to be integrated in a try/catch block to account for exceptions.

- 1.) Create a `URL` object which references to the home of the applet


```
URL currentPage = getCodeBase();
String protocol = currentPage.getProtocol();
String host = currentPage.getHost();
int port = currentPage.getPort();
String urlSuffix = "/servlet/SimpleServlet";
URL dataURL = new URL(protocol, host, port, urlSuffix);
```
- 2.) Create a `URLConnection` object


```
URLConnection connection = dataURL.openConnection();
```
- 3.) Prohibit the web browser to cache the URL data


```
connection.setUseCaches(false);
```
- 4.) Disclaim the system to allow read and write of data


```
connection.setDoOutput(true);
```
- 5.) Create a `ByteArrayOutputStream` to buffer the data that should be sent to the servlet


```
ByteArrayOutputStream byteStream = new ByteArrayOutputStream(512);
```
- 6.) Connect the `ByteArrayOutputStream` with the output stream `PrintWriter`

```
PrintWriter out = new PrintWriter(byteStream, true);
```
- 7.) Put the data for the server in the buffer


```
String data = "param1=" + URLEncoder.encode(value1)
              + "&param2=" + URLEncoder.encode(value2);
out.print(data);
out.flush();
```
- 8.) Set the *Content-Length*-Header


```
connection.setRequestProperty("Content-Length",
                              String.valueOf(byteStream.size()));
```
- 9.) Set the *Content-Type*-Header


```
connection.setRequestProperty("Content-Type",
                              "application/x-www-form-urlencoded");
```
- 10.) Send the data to the servlet


```
byteStream.writeTo(connection.getOutputStream());
```

The servlet sends answers through a response object that connects itself to the connection object that was instantiated for the request. Hence the applet needs to set up an input stream on this connections in order to receive the answer of the servlet. The following steps are necessary for this task.

- 11.) Create an `ObjectInputStream`
`ObjectInputStream in = new ObjectInputStream(connection.getInputStream());`
- 12.) Read data (*String*) with the method *readObject*
`String in = (String)in.readObject();`
- 13.) Close input stream
`in.close();`

In order to implement the applet, we made only minor extensions to APNNed to implement the communication procedures above and to avoid any kind of access to the local file system. It remains to describe the servlet side of the communication.

4 Responding to client request by a servlet

The servlet needs to take care of incoming requests by generating a new thread for each request. In this thread, the servlet must decode the message and interpret its content to trigger appropriate actions for a response. A servlet extends the class `HttpServlet`, its most relevant methods are *init*, *service*, *doGet*, and *doPost*. The last three functions have two parameters, a request object of class `HttpServletRequest` and a response object of class `HttpServletResponse`. The request object allows to access incoming HTTP-headers or data by opening an input stream. The response object allows to send HTTP-status codes and data by opening an output stream. Due to the similarities between receiving and sending, we describe only how to send data to an applet, the servlet opens an output stream using the response object. The servlet must only realize 4 steps to send serialized data to an applet.

- 1.) Set the content type for sending a serialized data object
`String contentType = "application/x-java-serialized-object";`
`response.setContentType(contentType);`
- 2.) Create an `ObjectOutputStream`
`ObjectOutputStream out = new ObjectOutputStream(response.getOutputStream());`
- 3.) Write data with *writeObject* into the output stream
`out.writeObject(new String("<APNN description>"));`
- 4.) The stream must be vacated. This ensures that the client get the complete data
`out.flush();`

Clearly, in addition to communication, the servlet implementation causes more work and code to establish the required functionality. Especially the control of the operations needs more considerations, for instance, the selection of analysis methods, its start, termination detection and return of results since the different analysis tools of the APNN toolbox are independent executables that are implemented in C and perform as independent processes. A sophisticated servlet implementation needs to take care of security, privacy, sharing among different users and applets; it need to take care of the usage of disc space, memory and cpu to avoid misuse and to possibly report on failures.

5 Conclusion

We briefly present the internet version of APNNed which supports loading, storing, and analysis of Petri nets via an HTTP server, so that APNNed can run as an applet in a general purpose web browser. Instead of describing the rather unchanged usage and appearance of APNNed, we describe how the necessary applet-servlet communication (HTTP-tunneling) can be realized in Java. Clearly, the current implementation covers a lot more details, for instance, transferring hierarchical nets implies to send a set of nets and information on its connectivity instead of a single net description. The current status of APNNed and the APNN toolbox is available at [1]. Ongoing work considers asynchronous interaction with time consuming analysis tools and further extensions to allow for a secure usage in an open environment.

References

- [1] <http://ls4-www.cs.uni-dortmund.de/qm/werkzeuge.html>.
- [2] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. *Quantitative Evaluation of Comp. and Comm. Sys.*, pages 356–359, Springer LNCS 1469, 1998.
- [3] F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri net notation. *Petri Net Newsletter*, 49:9–27, 1995.
- [4] P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. 7th Int. Workshop Petri Nets and Performance Models (PNPM'97), St-Malo (France), June 1997*, pages 81–90. IEEE CS Press, 1997.
- [5] P. Buchholz and P. Kemper. Efficient computation and representation of large reachability sets for composed automata. *Forschungsbericht 705*, Fachbereich Informatik, Universität Dortmund (Germany), 1999.
- [6] P. Buchholz and P. Kemper. Modular state level analysis of distributed systems - techniques and tool support. In W.R. Cleaveland, editor, *Proc. 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*. Springer, LNCS 1579, 1999.
- [7] P. Buchholz, P. Kemper, and APNNed group. APNNed - a net editor and debugger within the APNN Toolbox. In J. Desel, P. Kemper, E. Kindler, and A. Oberweis, editors, *5. Workshop Algorithmen und Werkzeuge für Petri Netze*, Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 694:19–24, 1998.
- [8] David Flanagan. *Java in a nutshell: A Desktop Quick Reference*. O'Reilly, 3 edition, Nov 1999.
- [9] David Flanagan, Jim Farley, William Crawford, and Kristopher Magnusson. *Java Enterprise in a nutshell*. O'Reilly, May 1999.
- [10] Marty Hall. *Core Servlets and Java Server Pages*. Prentice Hall, June 2000.
- [11] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep 1996.
- [12] P. Kemper and R. Lübeck. Model checking based on kronecker algebra. *Forschungsbericht 669*, Fachbereich Informatik, Universität Dortmund (Germany), 1998.
- [13] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*, Informatik Fachberichte 52, 1982.

Lightweight, Dynamic and Programmable Virtual Private Networks

Rebecca Isaacs
Computer Laboratory,
University of Cambridge
Cambridge, UK
Rebecca.Isaacs@cl.cam.ac.uk

Best Available Copy

Abstract—A Virtual Private Network (VPN) that exists over a public network infrastructure like the internet is both cheaper and more flexible than a network comprising dedicated semi-permanent links such as leased-lines. In contrast to leased-line private networks, the topology of such a VPN can be altered on-the-fly, and its lightweight nature means that creation and modification can take place over very short timescales.

To a programmable networking environment, such VPNs can be enhanced with fine-grained customer control right down to the level of the physical network resources, allowing a VPN to be employed for almost any conceivable network service. This paper examines some of the issues present in the provision of programmable VPNs. In particular, automated VPN “design” is considered, that is, how a VPN description can be translated to a set of real physical resources that meets customer requirements while also satisfying the goals of the VPN Service Provider (VSP). This problem—the distribution of resource allocations across network nodes in an optimal manner—has relevance for other approaches to VPN provision such as differentiated services in the internet [1].

The work described in this paper was carried out using a programmable networks infrastructure based on the switchlets mechanism [2]. It shows that automated VPN creation resulting in a guaranteed resource allocation is a feasible procedure that works well for both the VSP and for the customer that has requested a VPN. The problems inherent in dynamic VPN reconfiguration are also briefly explored together with the methods by which these might be addressed.

I. INTRODUCTION

Until recently, the management and control functionality of networks has been tightly coupled to the network hardware, leading to a “closed” and restrictive environment. The provision of new network services, signalling protocols and other software has been exclusively the domain of network equipment manufacturers, often, but not always, under the auspices of international standards bodies. The drawbacks of this approach are widely recognised:

- Although the use of standards ensures interoperability, the process of defining them is a slow and often highly political exercise. It is not unusual that by the time a standard is produced, technological developments have made it largely obsolete.
- Due to their monolithic nature, standards can be heavyweight and unwieldy, further stifling the impetus to develop and quickly deploy new network services.
- Mechanisms for introducing innovative network services are invariably very limited. If the demand has not already been anticipated, it is unlikely the network will be able to cater for the service in the optimal manner, if at all.
- Close integration of the network control software and the internal network elements makes upgrades and bug fixes difficult and costly.

Many illustrations of these shortcomings can be cited with respect to commonly deployed networks. Examples include the awkward integration of intelligent network services into the

telephone network, and the requirement for some ten thousand lines of code on every switch for UNI/NNI signalling in ATM networks.

A. Programmable Networking

Programmable networking has been advocated as a means of addressing these issues. The main idea of this approach is that the network can be programmed, in other words its physical resources accessed and manipulated, through an open programmable interface. Such an interface allows control and management functionality to be devolved from the internal network hardware to external processors, hence separating the role of equipment vendors from that of software and service providers.

Potential advantages of programmable networking include the opening up of the network to third parties, the easy introduction of sophisticated and hitherto unanticipated network services, and significant speedups in the deployment of such services. However, in practice these benefits are hard to deliver and pose some challenging questions such as how programmable network interfaces should be defined, how much abstraction is called for, as well as performance, robustness and security issues. Many of these issues are under active consideration in the research community [3].

B. Virtual Private Networks

Virtual Private Networks (VPNs) that make use of existing (possibly public) network infrastructure are a way of procuring the equivalent of a private leased-line network that is relatively cheap and flexible. Permanently dedicated resources are not required, and the topology and size of the VPN can be altered as required. These VPNs are most commonly deployed on the internet, often to provide connectivity between corporate LANs, or to enable individual mobile users to access a private LAN. The establishment of internet VPNs does not require large overheads: tunnelling is used to provide routing and addressing, and security measures such as encryption of the VPN's traffic and authentication protocols prevent data tampering and unauthorised access.

Issues of network performance and guaranteed quality of service, as well as mechanisms for pricing and charging where appropriate (especially when multiple ISPs participate at the boundary of the VPN), have not as yet been fully addressed, although their importance is recognised. Current VPN product offerings either operate solely on IP networks, or over a small number of other protocols.

The provision of VPNs in a programmable network environment takes the VPN concept a step further. It contains the means for a VPN to control not just its routing and addressing mechanisms, but any aspect of the underlying network resources that can be "programmed". Thus a VPN is not constrained to use a particular networking protocol.

Depending on the programmable interface used, the exposure of the network internals can guarantee VPN performance, and provide a low-level monitoring and charging mechanism for the VPN Service Provider (VSP). This scenario is possible when there is a means to isolate one VPN from another, so that each VPN can have exclusive control of and access to "its" resources. The switchlets concept [2] does exactly this by partitioning the physical resources and policing those partitions. An open control interface allows a VPN to access its partition without interference from any other VPN.

In general, resource partitioning combined with open network control has a number of benefits. Including:

- the removal of the need for a single instantiation of a prescribed control system (i.e. signalling mechanism or networking protocol)—each partition of the network can be controlled by a different system;
- the support for true multi-service networks where each type of service can operate in its appropriate environment, unaffected by other users of the network;
- the potential for a network operator to offer a *VPN Service*, in which VPNs comprising some subset of the available physical resource (in terms of both topology and capacity), can be acquired on demand and their allocated physical resources manipulated at will.

A VPN deployed in this environment can be extremely lightweight. Its existence could be as brief as a matter of minutes, and the network control software as minimal as required. An example is a VPN created purely for the duration of a video conference, running *service specific* control software tuned to support the requirements of video conferencing [4]. During the lifetime of such a VPN, its resource requirements might change, for example as a result of participants joining or leaving the conference, and this can be reflected in corresponding changes in its underlying physical topology and resource allocation.

This paper examines some of the issues involved in the provision of programmable VPNs, from the point of view of both the VSP and the VPN customer. These issues include:

- The tradeoff between expressiveness and simplicity for a VPN specification language that must cater for descriptions that range from the minimal to the comprehensive.
- The conflict between VSP goals of maximising resource usage, and customer demands for the best VPN to meet their needs.
- The theoretical intractability of finding an optimal VPN topology given these conflicting requirements.

The feasibility of automated VPN creation has been investigated by observing how VPN topology search performs with a

naïve implementation, and to what extent a simple heuristic improves performance. The scenario of a VSP using the switchlets infrastructure provides the context and the motivation for this work and is described in Section II. Section III proposes an approach to automated VPN design, encompassing a specification mechanism that reconciles the goals of the two parties to produce a cost function that is subsequently used to compute a VPN topology. Section IV presents the implementation experience, and further work concerning the problems inherent in dynamic reconfiguration of VPNs is discussed in Section V.

II. CONTEXT

A. Infrastructure

The infrastructure over which dynamic VPNs are provided is based on open signalling concepts. Control of the physical network is, as far as possible, devolved from the internal elements to general-purpose workstations. The functionality of the switch or router is encapsulated in an open control interface—typical operations available through such an interface include connection management, routing, alarm notification and the gathering of statistics. Examples of open switch control interfaces include GSMP [5] and VSI [6]. We use an interface developed in the Computer Laboratory called Ariel [7], [8].

Partitioning of the physical network is achieved by subdividing the resources of individual switches into one or more logically separate *switchlets*. Each of these is presented to its owning control system through an open switch control interface, which gives the illusion that the control system is managing an entire switch. This is the crucial feature that allows multiple control systems to co-exist on a single physical network, whilst also giving them fine-grained control of the resources they have been allocated.

A process called the *Divider* manages the creation, deletion and modification of switchlets. The Divider runs on a general-purpose workstation, ideally in a resource-controlled environment such as that provided by the Nemesis operating system [9]. Invocations on switchlet interfaces are policed on the control path by a Divider (one per switch) to ensure there is no interference between control systems. Fig. 1 shows multiple control systems sharing the resources of the physical network. Each control system makes invocations on the Ariel interface exported by the Divider for the corresponding resource allocation on that switch.

A full-blown virtual network can be constructed by acquiring switchlets in one or more network nodes. The *Network Builder* is responsible for the creation and deletion of virtual networks, and for allocating the resources required by those virtual networks. The topology of a virtual network need not map directly onto the underlying physical network. A single-link virtual network can span multiple physical links simply by reserving resources on all intervening nodes without exporting Ariel interfaces—such resource reservations are termed *tunnel switchlets*.

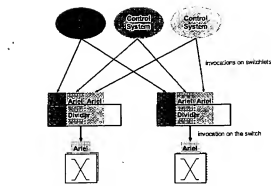


Fig. 1. Control systems operating simultaneously.

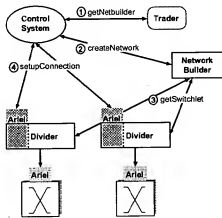


Fig. 2. Acquisition of a virtual network.

Communication between these architectural components takes place using a DPE, which runs over its own virtual network. A bootstrap virtual network is employed at start of day.

Fig. 2 illustrates the steps involved in the acquisition of a virtual network by a control system: a control system locates the Network Builder using a trader, and then asks it for a new network comprising some specified resources. The Network Builder in turn locates the Divider at each of the constituent nodes, and requests a new switchlet at each. After creating a switchlet, the Divider returns a new Ariel interface for that switchlet to the Network Builder. The switchlet interfaces are then passed back to the control system, which can then make invocations on switchlets directly and hence control its partition of the physical network resources.

The operation of control systems across possibly non-cooperating intervening networks is also addressed within the infrastructure [10]. Virtual networks can be set up that span multiple domains, with the result that a control system need not necessarily be aware of the underlying administrative boundaries. The infrastructure will take care of the tunnelling of traffic across other types of network, albeit with a potential loss of

service guarantees.

The provision for multiple control systems to operate simultaneously means that no single system need be prescribed for all users (which is not to say that multiple instances of the same control system can not co-exist). Although for many users standard, general-purpose control systems will suffice, others are able to run service-specific control systems tailored to their individual needs, if they wish to do so.

An implementation of this control framework is currently operational on an ATM network consisting of 5 ATM switches, 5 host workstations and 7 audio/video codecs. Ongoing developments include novel and innovative control systems, federated virtual networks over wide-area links, investigation into pricing and charging mechanisms, dynamic resource reallocation and adaptive control systems, and control system interoperability. For a general overview and more information see [7] and [8].

B. Related Work

Virtual networks within a programmable networking environment have been explored from two different angles. In *active networks* [11] programs can be inserted into the routers or switches and then executed on the messages passing through those nodes. This form of active network is the most extreme form of network programmability, but because the control and data paths are not distinguished it is difficult to provide hard guarantees for differentiation of services.

In contrast, the alternative approaches, which include the switchlets approach, partition network resources among virtual networks and provide some mechanism for independent operation of the virtual networks. Schemes such as Genesis [12] and VNRM [13] deploy a specific control system on a virtual network by instantiating objects that implement the desired control interface and protocol.

Using switchlets, no such built-in functionality is provided, but rather a handle onto a subset of the real, physical resources, which enables low-level and fine-grained manipulation of those resources. In consequence, a virtual network instantiated by means of a switchlet will be more lightweight and hence amenable to the automated VPN design that is the subject of this paper.

As an aside, direct manipulation of physical resources does not preclude the use of off-the-shelf control systems in a switchlets virtual network (indeed we run a cut-down version of IP on one of our virtual networks). It also does not mandate a particular resource abstraction for control, hence avoids needlessly restricting operations on the resource, or compromising efficiency, flexibility and performance.

Genesis addresses the automated deployment of virtual networks and their control systems through a process called spawning. This uses a network blueprint in the form of an executable profiling script which lists requirements for addressing, routing, management, topology, resources and so on. The profiling script is produced manually by a network architect, and the conflicting goals of VSP and virtual network customer that

are the subject of this paper are not addressed. Although Genesis has not as yet been implemented, the implementation plan described in [12] intends to tackle some of these issues.

Ongoing work concerning resource management in VPNs includes the "hose model" [14], in which a performance abstraction for IP-based VPNs is proposed that characterises the desired performance characteristics of a VPN by an aggregated capacity figure. Extensive evaluation using trace driven simulations shows that considerable benefit is gained by statistically multiplexing traffic across the VPN as a whole—a technique that can also be employed in the context of this work where appropriate because of the hard partitioning of resources between VPNs.

Proposals for differentiated services in the internet [1] effectively partition resources in a public network in a similar way. However, mechanisms for dynamically shifting resources from one aggregate traffic classification to another have not as yet been defined, although the desirability of such behaviour is recognised.

C. A VPN Service Provider

The nature of the issues considered in this paper is motivated by the envisaged requirements of a VPN Service Provider (VSP) making use of the infrastructure to sell dynamic and flexible VPNs to a range of customers. A VSP will benefit from the potential for efficient use of network resources, multiplexing gains inside the network, and flexible and fast response times to customer demands. The VSP may also offer extra value over the standard VPN service, for example with configurable reliability for VPNs, advance VPN reservations or the facility for dynamically loadable customer code inside a switchlet (i.e. extremely close to the physical network itself). A flexible and easy-to-use environment is needed in order to be able to introduce new services as desired.

The customers of a dynamic VPN service will include not only corporate users who currently employ VPNs over the internet or leased lines for their private network, but also companies selling network services themselves. These might be telephone companies, ISPs, off-the-street users needing a network for a short time, perhaps for a distributed game or a broadcast lecture, or even designers and developers of other network services. All these customers will have different needs over different timescales—some VPNs will be semi-permanent, while others rapidly and frequently change their resource requirements, or even only persist for a matter of minutes.

To fully realise the potential of the technology in this scenario, the VSP must offer a responsive, reliable and flexible service. Administrative overheads must be minimised; in particular the automated creation and modification of VPNs is paramount, not only to cope with large volumes of such requests, but also to be able to respond to them quickly and to cater for the many different types of customer. The remainder of this paper considers VPN specification and realisation in this context.

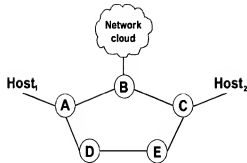


Fig. 3. Example network where preferred VPN topology from Host₁ to Host₂ is different for VSP and customer.

III. VPN DESIGN

So far this paper has described programmable VPNs and their advantages, as well as reviewing the switchlets concept and the prototype implementation of a VPN Service that provided the motivation for this work. The deployment of such a service on a real-world network is an attractive prospect, but in order to cater to the demands created by short-lived, lightweight and dynamic VPNs, automation of their provisioning is essential. The challenges of doing so are now investigated, with the overall aim of showing that in spite of some inherent difficulties, the automated design and deployment of VPNs is a feasible procedure.

The goal in VPN design is to create a virtual network that conforms to the specification provided by the customer, whilst maximising subsequent resource availability for the VSP. In order to increase the likelihood of being able to satisfy future customer requirements, the VSP should endeavour to spread the load of bandwidth, label space and switchlet allocation. As an example, Fig. 3 shows a network topology where although a customer might prefer that their VPN from Host 1 to Host 2 passes through the single node B, the VSP will route the VPN through D and E in order to maximise the local resource availability for VPNs originating within the network cloud.

This section considers the two main steps of the design process—customer specification of the desired VPN, and a means of mapping that specification to a near optimal topology and resource allocation.

A. VPN Description

The characteristics of a VPN that a customer may specify in a creation or modification request include:

- (virtual) topology;
- performance characteristics such as bandwidth, delay, loss tolerance, size of label space etc;
- temporal attributes i.e. start time and duration;
- cost;
- built-in extras, such as redundancy;
- contractual issues—penalties etc.

This type of specification, which describes both the desired VPN and specifies the guarantees made by the VSP with respect to that VPN, is often known as a *service level agreement* (SLA). Traditional SLAs used by WAN service providers are contracts between customer and network service provider that detail the level of service agreed in terms of measurable parameters. The content of an SLA usually covers type of service, data rate and QoS issues as well as contractual matters such as charging and compensation in the event of non-compliance with the agreement.

The automated translation of a VPN specification to a set of physical resources requires a formalised notation that is sufficiently expressive to capture a range of requirements, but is not unnecessarily restrictive. As pointed out in [8], VPN descriptions may range from the ill-defined (eg, "a cheap network between A and B"), to the comprehensively specified. Furthermore, the difficulties of predicting network traffic characteristics from a given source are well known. Insisting on a precise specification when the user does not know or understand their requirements in such detail only leads to sub-optimal network usage, either because the source exceeds its usage parameters and loses data, or else because the network is over-provisioned and resources wasted.

A VPN may not necessarily be double-ended, in other words a VPN specification which describes the origin and characteristics of the traffic without explicitly stating its destination should be valid. The conversion to a complete specification should be able to take into account potential optimisations resulting from this single-endedness to multiplex where possible.

The exact nature of the notation adopted depends to some extent on the capabilities of the underlying network and the degree of specification that the VSP wishes to allow its customers. Greater freedom leads inevitably to more complexity in the system. In our experimental environment we allow a fairly limited choice of VPN parameters, namely participating nodes, size of label space, bandwidth and maximum hop count. Other characteristics that could easily be incorporated include delay, jitter, duration, start-time, redundancy and so on.

After an SLA is defined, the partial VPN specification it contains must be converted to a complete specification, that is, one that represents an instantiation of the specified VPN on the actual physical networks. This involves an augmentation of the incomplete description such that all of the necessary physical topology, together with the resources required on each node and link, is explicitly specified.

Once a complete description is arrived at, a VPN can easily be expressed as a set of switchlet specifications. The mapping of a customer-provided VPN description to a complete description is the most complex step in the process of creating a VPN. Two questions must be addressed:

1. Does the described VPN make sense in terms of the actual physical topology—is the description *feasible*?
2. How do we arrive at an arrangement of VPN topology which:
 - satisfies the customer requirements,

- can be realised with available resources, and
- is optimal for the VSP?

Checking of feasibility is straightforward, as the VSP has global knowledge about the network topology. At this step it may also be able to refine the SLA so that it includes at least those nodes that *must* be present to meet the stated requirements. In contrast, the second question is quite difficult to answer, and is discussed at length in the next section.

B. VPN Routing

Determining an optimal route, or topology, of a VPN is a non-trivial problem. It can be expressed formally as follows:

The cost of a subgraph G' is determined by some function $f(G')$. Given a weighted graph G and a set of vertices in G denoted V' , what is the cheapest way of forming a connected subgraph containing V' according to the cost function f ?

This problem is similar to that of finding a minimum Steiner Tree, where the goal is to connect a set of vertices in the graph by finding a minimum-weight spanning tree that can also use any of the remaining vertices. The difference is that the aim is not to find a minimum spanning tree, but to find the cheapest subgraph according to a cost function that will vary from one VSP to another, and may even be altered over time at the same VSP.

As an example consider the networks shown in Fig. 4. The requested VPN consists of the nodes A , B and C , and the edge weights have been calculated as shown, according to the VPN resource description, current resource availability and local policy. A costing function that favours the minimum-weight VPN topology will produce the subgraph highlighted in the left-hand network, whereas the subgraph of the middle network is produced if minimising the number of links is given greater priority. If full redundancy together with minimum-weight is required, the subgraph highlighted in the right-hand network will be the result.

It is clear from the problem description that once a partial VPN specification has been derived from the given SLA, there are then three stages in reaching a solution:

B.1 Assign edge weights

The weight assigned to any individual edge will depend on:

- availability of the resources specified in the VPN description (either explicitly or implicitly) as required for that link;
- any arbitrary cost associated with a link, as determined by local policy.

A link that is unable to provide the required resources is immediately assigned a weight of infinity and removed from further consideration.

B.2 Generate a cost function

The cost of a VPN is determined by a combination of customer requirements—a candidate VPN that doesn't meet the specifications of the SLA will have infinite cost—and the circumstances of the individual VSP. For example, in some cases bandwidth may "cost" more than label space (perhaps because

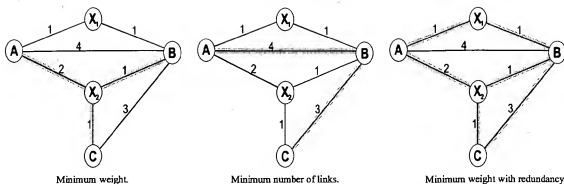


Fig. 4. Three cost functions producing different optimal VPN topologies.

the VSP has very few customers), whereas in other networks the reverse will be true. The costing function should reflect the relative priorities in the local domain, and balance the requirements for a particular VPN with the overall needs of the VSP.

A cost function routine is automatically generated by parsing the SLA to derive explicit rules and incorporating these in a template that includes rules dictated by local policy. The details of this procedure will vary from one VSP to another, but there is always commonality in that the cost calculation routine used at any one time is (probably) unique to the individual VPN under consideration.

B.3 Find the best VPN topology

As stated above, a VPN's topology will be determined by a combination of the edge weights and the cost function, according to local policy.

Intuitively, an implementation that always finds the best subgraph (i.e. VPN topology) will be computationally expensive because of the large number of candidate subgraphs. If we consider a subgraph cost function that is simply the sum of the constituent edge weights, and if cycles are not permitted in the subgraph, then the problem becomes that of finding the minimum Steiner Tree. This is known to be NP-complete [15], and thus so is the optimal VPN topology problem described here.

Nevertheless, using a combination of sensible heuristics and careful engineering, an implementation can be produced that gives tolerable performance for a network of reasonable size. In the following section the brute-force solution is analysed in order to derive heuristics that give close to optimal solutions, as well as making the problem tractable within the desired bounds. These bounds are determined by deciding what "tolerable" performance is, and how large a "reasonable" size of network should be. Experimental observation supports the claim that the performance of the resulting algorithm is adequate.

IV. IMPLEMENTATION EXPERIENCE

This section examines in some detail the implementation of the search for an optimal VPN topology. By means of experimental results we compare the performance of the naive approach with that possible using a simple heuristic.

A. Algorithm

A brute force method of determining the optimal VPN topology is to construct every possible connected subgraph that satisfies the given VPN description, calculate the cost of each and then choose the cheapest. The pseudo-code of Fig. 5 describes an algorithm that takes this approach. For clarity, finer points of the implementation such as finding optimal topologies that incorporate redundant links are ignored.

Let V' denote the set of nodes in the VPN. For any $v \in V'$, $\text{single-hops}[v]$ contains all paths from v to each of the other nodes in V' , that do not pass through any other node in V' and do not contain any cycles. The function CALCULATE-COST is generated as described in Section III-B.2.

The construction of single-hops is straightforward using a modified form of the Floyd-Warshall transitive closure algorithm to generate one possible set of paths between VPN nodes, and then taking the transitive closure of the resulting paths to obtain all possible paths. At this stage paths with edges that have infinite weight (i.e. cannot fulfill the resource requirements) are discarded.

The algorithm to find the minimum-cost subgraph containing V' is shown in Fig. 5. At line 7 whichever VPN node was added last to the subgraph is taken, and its paths to other VPN nodes looked up in single-hops . The procedure is then called recursively for each of these paths. Note that the resulting collection of paths will not necessarily be disjoint, but a subgraph is a candidate (i.e. costed) only if it contains all of the nodes in the VPN. At line 8 processing is also carried out to ensure that continuously cycling paths are not considered, and to guarantee termination within a reasonable time period, the search is abandoned after a certain (large) number of recursions.

```

MIN-SUBGRAPH(subgraph)
1  if all nodes  $V'$  in subgraph then
2     $c = \text{CALCULATE-COST}(\text{subgraph})$ 
3    if  $c < \text{min-cost}$  then
4       $\text{min-cost} = c$ 
5       $\text{min-subgraph} = \text{subgraph}$ 
6  else
7     $\text{paths} = \text{single-hops}[\text{subgraph.last}]$ 
8    for each  $p$  in  $\text{paths}$  do
9       $\text{MIN-SUBGRAPH}(\text{subgraph} + p)$ 

```

Fig. 5. Brute-force algorithm to find a minimum-cost subgraph.

Let m be the number of edges in the graph G and let k denote the number of nodes in the VPN. The number of paths between any two nodes without cycles is at most 2^m , and for any node v there are paths to at most $k - 1$ other nodes. Therefore the maximum number of iterations of the loop at line 8 is at most $(k - 1)$, with each iteration recursing at most 2^m times. As expected, the brute-force algorithm is computationally intractable.

B. Heuristics

The intractability of this algorithm arises because the cost function is not monotonically increasing, i.e. the overall cost of a subgraph may reduce as it encompasses more of the VPN nodes (for example if redundancy is required by the customer). As stated above, if the cost function does happen to be cumulative, then the solution is the minimum spanning tree and can be found in polynomial time using any of the well known algorithms.

However, even with a non-monotonic cost function, heuristics can be applied to make the procedure practical for networks of reasonable size. At the potential expense of sub-optimal results, the search time can be speeded up by reducing the size of the search space. Two approaches are possible:

- abandon partial subgraphs that are thought likely not to be optimal in the future;
- order the search sequence to consider first those subgraphs that are most likely to be optimal and stop the search at the first hit.

The first approach can be implemented by choosing a reasonable cut-off point, and abandoning partial subgraphs that exceed this cost. This is obviously more likely to perform well if the cost function is 'almost' monotonic. Additionally, the cost function may contain cumulative aspects that can be considered in isolation. For example, partial subgraphs can be discarded on the basis of exceeding the desired hop count or end-to-end delay. However, a significant drawback with this approach is that the additional computation associated with calculating costs of partial subgraphs may dominate the running time and negate any improvements gained from the smaller number

of subgraphs considered in total.

In contrast, the second technique will tend not to return as good results, especially with denser graphs, but will spend significantly less time calculating partial subgraph costs. Another key advantage is that if a subgraph that meets the customer's requirements (with the possible exception of a cost constraint) exists, then it will always be found. The first technique runs the risk of not finding a solution, due to having abandoned that subgraph earlier, whereas with ordered search if a solution exists then it is guaranteed to be found eventually.

Thus with the ordered search heuristic the VSP may lose out on fulfilling its overall goals, and the customer may be penalised on VPN price, but both gain substantially from search speed increases. Of course in practice the VSP can subsequently mitigate any losses by modifying the cost function appropriately. The results in the following section support this analysis, and demonstrate that by incorporating the ordered search heuristic, automated VPN design is a feasible procedure.

C. Experimentation

This section presents experimental results to back up the assertion that in spite of its computational complexity, the process of automated VPN creation is practical on a reasonably large network.

For ease of implementation, an interpreted scripting language was used, hence the CPU usage timings give an idea of relative improvements rather than demonstrating what can be achieved absolutely. A well-engineered solution written with the appropriate tools would perform much better. The network topologies were generated using the TIERS random network topology simulator [16], with a small range of edge densities reflecting the sparsity generally found in real-life networks. The networks are assumed to be under the administrative control of a single VSP and are accordingly no larger than 25 nodes.

Because the topologies are produced randomly no two runs give the same results and there are occasional large fluctuations for relatively large and dense networks. Notwithstanding this, some care has been taken to ensure that the results included represent typical executions, and all data points are average values over 10 runs.

The graph in Fig. 6 shows how badly the brute-force algorithm actually performs in practice. It indicates the extreme deterioration in performance as both the graph size and the proportional VPN size increase. Once the VPN covers more than about 30% of the network, the search time increases dramatically.

Problems are also caused for brute-force searching by increases in graph density, and this is shown by the graph in Fig. 7. This graph shows the search time for a VPN covering a fixed proportion of the physical network—60%, with average node degree increasing from 1 to 4. Search times increase substantially once the graph size exceeds 10 nodes. Clearly the brute-force approach is not adequate for any network of reasonable size.

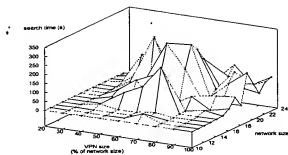


Fig. 6. Performance of brute-force search on sparse networks.

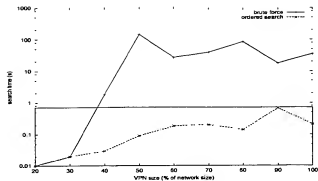


Fig. 8. Brute-force vs ordered search on a 20-node sparse network.

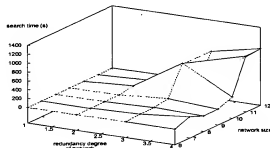


Fig. 7. Performance of brute-force search for VPN size 60% of network size.

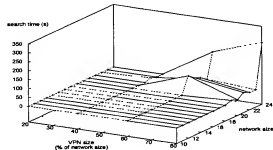


Fig. 9. Performance of the ordered search heuristic on sparse networks.

The next experiment examines the gains that can be made using the ordered search heuristic as well as the corresponding cost penalty. The heuristic is incorporated in the pseudo-code algorithm of Fig. 5 by ordering the paths list iterated over at line 8 according to the result of applying the cost function to each one. Recall that these are acyclic single-hop paths between 2 nodes of the VPN, therefore the overheads of the cost calculations are minimal.

The graph in Fig. 8 compares the search times of the brute-force algorithm and the ordered search algorithm for a medium-sized graph of 20 nodes. It shows the expected marked improvement in performance using the heuristic. Note that the y-axis is log scale.

Finally, the first two experiments run using brute-force searching are repeated with the ordered search heuristic. To facilitate comparison, the results are plotted with the same Z-axis range which confirms that the large humps present in the graphs of Figs. 6 and 7 only appear here for much bigger networks. As expected, ordered search performs much better in both cases.

V. FURTHER WORK

So far the issue of automated VPN creation has been addressed without regard to possible reconfigurations of the VPN in the future. In fact, the ability to alter the topology and resource allocation of a VPN on-the-fly is one of the main advantages of the VPN Service described in this paper. Some scenarios where dynamic reconfiguration might be used include:

- A service-specific control system tailored to a particular multicast application, for example video-conferencing, where changes to group membership may require switchlets to be created at additional nodes, or even for nodes to be removed from the VPN. Similarly, end-user requests for improvements in video quality may be met by increasing the bandwidth allocation of the VPN on the relevant links.
- A control system for a VPN supporting mobility that modifies its topology in response to the movement of wireless devices—holding on to resources only at adjacent base stations, and discarding those that are “far away” from the current location.
- An efficient IP-on-ATM control system where the size of the label space in the underlying switchlets is altered rapidly in response to the creation or termination of traffic flows.

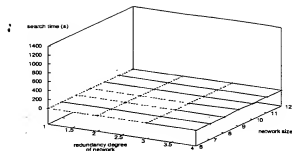


Fig. 10. Performance of the ordered search heuristic for VPN size 60% of network size.

VPNs that reconfigure as and when required can make more efficient use of network resources and are more flexible than statically pre-configured networks. The risk for the VPN owner is that a reconfiguration request may be rejected by the VSP, perhaps because of lack of resources, with the resulting loss for the VPN owner of efficiency, flexibility and possibly even the ability to provide a service to their own customers. The appropriate charging mechanisms to reflect this trade-off are the responsibility of the VSP.

Automated reconfiguration should be built in to the system for the same reasons that automated creation and deletion are necessary, as discussed in Section II-C. Reconfiguration can involve changes to either the VPN topology, or to its resource allocation, or both. The problem considered here concerns just alterations to VPN topology, which may of course result in changes to the overall VPN resource allocation as well. The details of in-place reconfiguration of resources at a particular switch are currently the subject of ongoing research.

A. Analogy With Dynamic Multicast Routing

The problems of dynamic VPN topology reconfiguration parallel those found in dynamic multicast routing (but not for datagram networks). The issues, which are extensively discussed in the literature (see, for example [17]) involve compromises between optimal placement of a node joining the multicast session and the corresponding deterioration of the tree as a whole.

As with VPN creation, the initiation of a multicast session generally involves a computation of the optimal topology. The topology of multicast trees, which are single source to multiple receivers, can be influenced by many different constraints. These include scalability, QoS requirements such as end-to-end delay bounds, efficiency requirements and algorithm complexity considerations. When a new destination is introduced into the multicast tree, naive (computationally cheap) placement of the new node can result in deterioration of the quality of the tree—with some chance of making the initial effort expended in calculating an optimal tree a waste of time. On the other

hand, constant reconfigurations of the topology to maintain an optimal, or close to optimal, tree are also expensive, and can disrupt traffic to existing members of the multicast group. Most solutions adopt a compromise where changes to the tree are kept as localised as possible, and periodically the entire tree is re-routed to try and maintain a close to optimal topology.

Some examples of current research in this area include [18] and [19]. In [18], an algorithm is presented that maintains a good (but not optimal) multicast tree (in terms of minimising the sum of the edge weights) without undue computational cost using Kruskal's shortest-path algorithm. Rearrangements are triggered after a certain amount of deterioration in the quality of the tree—determined by counting the number of changes in a vicinity—but enough state is maintained to be able to confine the necessary rearrangements to localised regions. In contrast, in [19] the quality of the multicast tree is assessed according to whether it meets delay variation constraints. However there is a similar emphasis on minimising the effects of leave and join operations on the tree as a whole.

B. Discussion

Although it has a lot in common with the multicast problem, dynamic reconfiguration in the context of VPNs in the switchlets environment has some important differences.

On the one hand, the allocation of physical network resources directly to a VPN owner means that topological rearrangements of the VPN at the whim of the VSP are not necessarily going to suit the way the customer is using their portion of the network. On the other, the difficulties can be eased by the fact that the customer may be able to explicitly identify “sensitive” regions of their network that should not be modified, and nodes where disruption is tolerated. Any threshold of deterioration (which will roughly translate to cost to the customer) can be chosen on a per-VPN basis, and to take this to an extreme, the mechanism by which the topology is updated can also be specified by the VPN customer. In effect this gives the VPN owner, i.e. the person paying for the network, a great deal of control, allowing them to tailor the dynamic reconfiguration behaviour as appropriate.

However, it must also be expected that many VPN customers will not be willing or able to specify such details. A minimal VPN specification such as “Give me a network containing nodes A, B and C” which at some point in the future is modified by “Add node D to my network” is a perfectly valid one. The question is whether this customer would be happy with a less than optimal topology where D is simply joined to A, B and C, or whether a rearrangement of all the links in the VPN at this point would be acceptable. In general it seems unlikely that the latter option would be preferred, and indeed should a customer require this they could always request a new network containing A, B, C and D at the appropriate time. On the other side of the coin, the VSP may have expended much effort in calculating the original network topology, and this effort may subsequently be rendered useless by the changes requested by the customer, especially if these changes occur frequently. It would be in the interests of

the VSP to have some sort of characterisation of the "dynamicity" of a VPN at the time of creation. A VPN that is likely to change a lot over its lifetime can be given a topology that is sub-optimal in terms of cost but is more resilient and gives a cheaper overall topology with plentiful changes.

In summary, the following aspects relating to reconfiguration can be specified by the VPN customer:

- the expected rate of membership modification;
- tolerance of disruption to the VPN as a whole;
- tolerance of disruption at particular nodes;
- any preferred means of rearrangement;
- thresholds for triggering rearrangement;
- amount prepared to pay.

A combination of these factors could be used to influence the choice of initial routing. A VPN that underestimates its degree of dynamicity will initially pay an extra cost for unanticipated reorganisation overheads. However it is also possible in this situation that the system could slowly adapt to the observed behaviour of a misbehaving VPN, and successively produce less optimal but more resilient topologies.

The nature of a VPN topology that is resilient to change, in terms of maintaining its overall cost as its membership alters, will vary according to the cost function in use by the VSP. Algorithms proposed for dynamic multicast routing, such as ARIES [18], could be adapted to operate in this environment according to the chosen cost function. Investigation into this aspect of the provision of dynamic VPNs is continuing.

VI. CONCLUSION

The automation of VPN design and deployment has many advantages for both VSP and customer. It allows a customer to obtain a VPN extremely quickly, while still being able to explicitly specify the VPN's performance and other characteristics. The administrative overheads for the VSP are greatly reduced, and network usage efficiency is enhanced. With the incorporation of charging mechanisms correlated against computed VPN cost, the administrative burden can easily be further minimised.

This paper has shown that in spite of computational complexities it is feasible to automate VPN topology generation in accordance with both customer requirements for the VPN itself and VSP goals for the network as a whole. A VPN service level agreement, together with current resource allocation and global policy, can be incorporated in a cost function that is then used to determine the optimal physical topology of a VPN that satisfies the specification. The generation of this topology is made computationally tractable by means of an ordered search heuristic. Experiments have demonstrated that even within a non-optimised environment, a VPN can generally be determined in under 2 minutes on a sparse, reasonably sized network of up to about 25 nodes.

A programmable network infrastructure, such as that facilitated by switchlets, opens up the way in which a VPN can be exploited. As well as a flexible and dynamic network topology and resource allocation, there are no built-in restrictions on the

control system, network protocol or end-user applications using that network. This "open" environment, with lightweight and dynamic VPNs, is a practical realisation of "networks-on-demand".

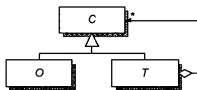
REFERENCES

- [1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, December 1998.
- [2] Jacobus E. van der Merwe and Ian Leslie. Switchlets and dynamic virtual ATM networks. In Aurel Lazar, Roberto Sarneco, and Rolf Stadler, editors, *Integrated Network Management V*, pages 355–368. IFIP & IEEE, Chapman & Hall, May 1997.
- [3] Opensig working group. Details at <http://comet.columbia.edu/opensig/>.
- [4] Jacobus E. van der Merwe and Ian M. Leslie. Service specific control architectures for ATM. *IEEE Journal on Selected Areas in Communication*, 16(3):424–436, April 1998.
- [5] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching, T. Lyon, and G. Minshall. Ipsilon's General Switch Management Protocol specification version 2.0. RFC 2297, March 1998.
- [6] William P. Buckley. Virtual Switch Interface (VSI) implementation agreement. Available from <http://www.msforum.org/>, November 1998.
- [7] Sean Rooney, Jacobus E. van der Merwe, Simon A. Crosby, and Ian M. Leslie. The Tempest: A framework for safe, resource-assured programmable networks. *IEEE Communications Magazine*, 36(10):42–53, October 1998.
- [8] Jacobus E. van der Merwe, Sean Rooney, Ian Leslie, and Simon Crosby. The Tempest—a practical framework for network programmability. *IEEE Network Magazine*, 12(3):20–28, May 1998.
- [9] Ian Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communication*, 14(7):1280–1297, September 1996.
- [10] Herbert Bos. Application-specific policies: Beyond the domain boundaries. In Morris Sloman, Subrata Mazumdar, and Emil Lupo, editors, *Integrated Network Management VI*, pages 827–840. Boston, May 1999. IFIP & IEEE, Chapman & Hall.
- [11] David L. Tenenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Weisenthal, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.
- [12] Andrew T. Campbell, Michael E. Koumavis, Daniel A. Vilella, John B. Vicente, Hermann G. De Meer, Kazuo Miki, and Kalai S. Kalaiachavan. Spawning networks. *IEEE Network Magazine*, 13(4):16–29, July/August 1999.
- [13] Andrew Do-Sung Jun and Alberto Leon-Garcia. Virtual network resources management: A divide-and-conquer approach for the control of future networks. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 98)*, Sydney, Australia, 1998.
- [14] N.G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K.K. Ramakrishnan, and Jacobus E. van der Merwe. A flexible model for resource management in virtual private networks. *Computer Communication Review*, 29(4):95–108, October 1999. *Proceedings of SIGCOMM* September 1999.
- [15] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [16] Matthew B. Doar. A better model for generating test networks. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 96)*, pages 86–93, London, UK, November 1996. Source code available from <ftp://ftp.nexen.com/pub/papers/tiers1.2.tar.gz>.
- [17] Matthew Doar and Ian Leslie. How bad is naive multicast routing? In *IEEE INFOCOM'93*, pages 82–89, San Francisco, USA, April 1993.
- [18] Fred Bauer and Anujan Varma. ARIES: A reconfigurable inexpensive edge-based on-line Steiner algorithm. *IEEE Journal on Selected Areas in Communication*, 15(3):382–397, April 1998.
- [19] George N. Rouskas and Ilia Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communication*, 15(3):346–356, April 1998.

Best Available Copy

***Experiences from extending a legacy
system with CORBA components***

COT/4-08-V1.3



Centre for Object Technology

*Centre for
Object Technology*

Revision history:	V1.0	1999-03-01	First draft.
	V1.1	1999-05-18	Second draft to review.
	V1.2	1999-06-21	Third draft to review.
	V1.3	1999-10-01	Final version.

Author(s): Allan R. Lassen, RAMBØLL
Jacob Steen Due, RAMBØLL
Emil Hahn Pedersen, RAMBØLL
Mohammad Al-Shamri, RAMBØLL

Status: Final

Publication: Public

Summary:

A presentation of the experiences gained by performing an extension of a legacy system by using object-oriented technology. The experiences are the basis of a general approach to extend a legacy system with add-on functionality based on the CORBA technology.

© Copyright 1999

The Centre for Object Technology (COT) is a three year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Danish National Centre for IT-Research (CIT) and the Danish Ministry of Industry.

Participants are:
Maersk Line, Maersk Training Centre, Bang & Olufsen, WM-data, Ramboll, Danfoss, Systematic Software Engineering, Odense Steel Shipyard, A.P. Møller, University of Aarhus, Odense University, University of Copenhagen, Danish Technological Institute and Danish Maritime Institute

1. Introduction

In the IT industry many systems, developed a long time ago, are still maintained because they are crucial for the IT business. Introducing new and better technology in the development process raises a challenge on how to adopt the new technology in the older systems. In this document we present our experiences of extending a system using object-oriented technology and CORBA. For more information on our project please refer to [3] which contains the design of the new system.

2. Goal of project

The goals of this project were first of all to gain experience and secondly to produce a final product. The main factors were:

- We saw a need to develop a method that allows new functionality to be added to older legacy¹ systems.
 - The new functionality should be in the form of add-on modules.
 - The changes needed in the legacy system should be minimised (and ideally there should be no changes at all in the legacy system).
- The architecture should allow implementation in multi-tier architectures.
 - We wanted to implement it in CORBA using the latest version of Oracle Application Server (OAS). This was basically a commercial reason. We had good experience with other Oracle tools and we expected that OAS would integrate nicely with those Oracle databases that many of our systems use.

The project is part of the research conducted in the Centre for Object Technology [1]. The actual project is part of industrial case 4 concerning integration with non-OO systems. By using object-oriented technology a partial goal of the project was to examine if the above-mentioned requirements could be realised successfully.

The basis of the project was a workflow and document management system, WorkSAFE, developed by RAMBØLL [2] as a standard system. Although the system is relatively new and well documented it corresponds to the definition of legacy system used in this report.

The current workflow system is designed to be used internally in an organisation. We wanted to give external partners access to selected parts of the system (i.e. access to some of the documents and to participate in the workflow for quality assurance). Partners should learn the system quickly. Eventhough the current system has a very general user interface, it takes time to learn and we wanted to supply a user interface designed to meet the requirements of the partners.

¹ In this report we use the term 'Legacy system' to denote monolithic systems. Typical examples are custom developed systems for various purposes such as accounting and other administrative purposes. Such systems are traditionally difficult and expensive to develop and maintain.

Whereas this may seem as a rather trivial case it does in fact raise a number of interesting questions that have turned out to produce implications in general when adding new functionality to legacy systems:

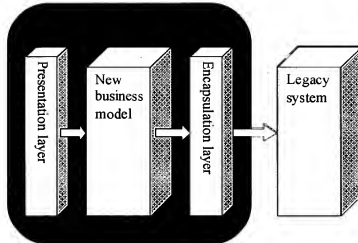
- How do we get a maintainable interface to the legacy system?
- How do we reduce our dependency on changes in the legacy system?
- How does the choice of technology influence our implementation?

3. Architecture

A key issue in adding new functionality to the legacy system, is the architecture to be used for the add-on modules. You may ask yourself the question: What kinds of interfaces are available for the legacy system?

Many legacy systems are using a relational database as persistent storage. The database tables implement an entity-relationship model, and usually you can easily transform it into an object model by representing each entity as a class. However, in our case the system had a complicated E/R-model because it used a meta-model of all its data. Therefore we decided to use an interface based on its Oracle database views and stored procedures. In another system you may have other alternatives, such as a documented API.

In our project we came up with a logical architecture, which we in general find useful. The architecture is illustrated in the figure below.



The encapsulation layer contains the classes corresponding directly to the legacy system interface. The layer is (and must be) as simple as possible and contains only the functionality that exists in the legacy system. The purpose of this layer is to shield the new business model from the complexities and idiosyncrasies in the actual implementation of the legacy system. Therefore it is very important to make the layer as simple as possible.

The new business model is a model of the relevant parts of the legacy system and contains classes with business logic of the new functionality. The classes use the encapsulation layer to perform actions towards the legacy system. The business classes serve as an abstraction of the legacy system for the user interface and they only contain information that is relevant to the new clients.

In a traditional multi-tier architecture there is a database tier and one or more business application tiers. In our architecture the business logic is performed in a combination of the new business model and the legacy system.

The presentation layer implements the user interface like in traditional multi-tier architectures.

3.1 Considerations about the clients

In our project the new functionality did not depend on the existing user interface and we considered three solutions to the presentation layer:

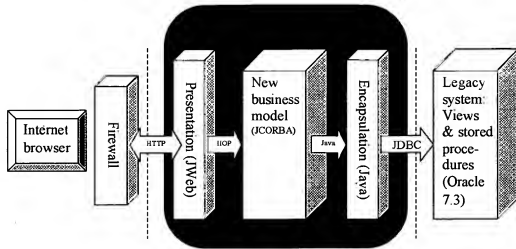
- An HTML based presentation where HTML pages are generated from the presentation layer on a server and displayed in an Internet browser on the client.
- A presentation using Java applets running within the Internet browser. The applets are Java classes that will be loaded from a server by the browser and then run on the client. The applets will communicate through CORBA IIOP with business classes running on a server.
- A stand-alone client application e.g. developed in Java. The application does not require a browser and will communicate with business classes in the same way as Java applets. Some mechanism to install the application would be needed.

We believe the Java applets or application would offer the best facilities for implementing a nice and fancy user interface. However to have our end-users, i.e. a partner, accessing the system through the Internet raises a problem with firewalls. Currently most firewalls do not allow the CORBA IIOP protocol to pass through. Eventhough, we get our company firewall to work with IIOP the user will probably sit behind a partner firewall that we not are able to control. Therefore we decided to dynamically generate HTML from our presentation layer.

An alternative is to implement a Java based user interface by marshalling and tunnelling the Java objects through the HTTP protocol. But it requires some extra work to implement and we didn't want to struggle with this issue in our project. Hopefully the problem about getting the IIOP protocol through firewalls will soon be solved and thus make implementation of fancy user interfaces easier.

3.2 Used technology

The Oracle Application Server 4.0 is the platform of our implementation of the architecture and we use techniques from the product as illustrated below:



The figure shows a physical model of our new system. All the classes in the system are implemented in Java. The business classes are subclasses of a JCORBA superclass, which is a class that Oracle has derived from the standard OMG CORBA class. JWeb classes are classes that dynamically generate HTML pages from Java. The dotted, vertical lines separate the parts that can be placed on separate physical servers.

4. Development process

4.1 Modelling the problem domain

The architecture described in the previous section naturally leads to the following steps in the modelling.

- An external model of the legacy system must be created in order to provide the encapsulation layer.
- A new business model is made.
- The encapsulation and business models are mapped together.

In our case it was relatively easy to construct the encapsulation layer because of the interfaces available to the legacy system, but this is not always the case. In other systems the nature and structure of the legacy system may make the task more complicated, e.g. if the only way to enter and query information in the legacy system is through its user interface. Though the goal is to use existing code from the legacy system there is a risk on having to establish some of the functionality in the encapsulation layer.

The business model could be created with only little awareness of the old legacy system and with focus on the application field. In effect this model was created in the same way as creating a model for a completely new system. Of course the model has a constraint that it must conform to the legacy model.

Interestingly enough, we found out that even though the developers with real knowledge of the legacy system found it relatively easy to create the encapsulation layer, they also found it demanding to create the business model due to the difficulty of abstracting from the familiar terms used in the legacy system. Actually, the participants with superficial knowledge of the legacy system were the ones who made the greatest progress on the business model. In practice two tracks were formed to develop the layers in parallel and this worked surprisingly well because of the clear division of the responsibilities in the layers.

In our case the business model did not introduce the need for extra information - it could all be stored in the legacy system. Therefore the problems associated to such a need have not been an issue. If such a need occurs the correct place for storing the information is in the business model when it is required that the legacy system shall not be modified. However, the complexities in securing transactional control, data security and performance are not described in this project, eventually an exercise for the reader ☺.

The objects in the business model may be distributed when they are implemented as CORBA objects. This raises an interesting opportunity to obtain better fault tolerance in the system. However we didn't investigate this topic further in our project.

The mapping between the encapsulation and business models proved to be relatively easy due to several reasons. One reason is the major factor that the problem domain for the new functionality was close to the problem domain of the old legacy system. This must be assumed to be the case in most situations where add-on functionality is put on top of legacy systems. Honestly, our legacy system was developed with OO techniques, as an implementation of an OO model in the relational paradigm. Though this implementation probably reduced the complexity in mapping the models we still believe our experience is correct because we interface to the legacy system as a traditional system implemented in a relational database.

4.2 Functionality

In developing the object structure of the business model and methods we had good experiences with small descriptions of scenarios². The scenarios illustrated the work processes we wanted to support and they were combined with the involved dialog flows. The scenarios and dialog flows helped to identify key methods and events in the system.

² The scenarios descriptions could have been replaced by the standardised use cases from UML but we chose to use descriptions that were less formal.

4.3 Design

After creating a good business model we found it difficult to describe a detailed design without real knowledge on the implementation platform. It is a general challenge when you take new technology in use. We believe it is a good idea to try some experiments and perhaps find some examples to benefit from. Our project described in [3] may serve as an example. It is also a good idea to see if there is a design pattern describing a solution to similar problems. An introduction to patterns is given in [4].

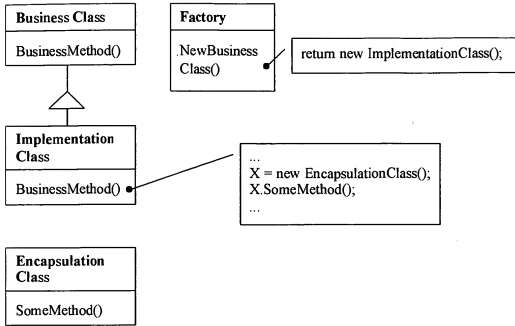
One of our concerns was how the clients should connect to the system and how we could keep track of the connection between user interactions. We found a good solution by introducing a session class and a session manager that handles the user connections (see [3] for details).

Our object-oriented system uses the relational database in the legacy system as persistent storage. It raises a question on how to manage the information when it is represented both in the relational database itself and in the object instances that we create in our system. We wanted to be certain that the information was updated correctly in our system as well as in the legacy system. Indeed, it is a problem if the legacy system changes the object values after our system have loaded them into an object instance or vice versa. We were lucky that the legacy system had implemented a locking strategy we could use to notify when we were going to update the information. In another system you may have to struggle with this issue.

In order to handle the information in the legacy system we used the following two general principles:

- We introduced an object manager class for each class that would be manipulated in our new system. The manager implemented methods to create, retrieve and store objects while ensuring the integrity between the two systems. The manager methods also checked and ensured that a particular object was only retrieved once. A manager class must only be instantiated once and we ensured this by using the Singleton pattern [5].
- Classes that could not be updated in our system were implemented as data banks. The data banks provided methods for looking up information in the legacy system. Persons and companies were not administrated by our system and we thus implemented a data bank for persons and a data bank for companies.

The mapping between business classes and encapsulation classes leads to high coupling between our new functionality and the legacy system. We solved this problem by using a Factory Method pattern [5]. Our business classes describe the general interface used by the presentation layer and they only implement general methods. Methods that depend on encapsulation classes are implemented in derived implementation classes. The principle is illustrated in the figure below:



This technique proved to be very useful. In principle, we are able to transform our system to use another equivalent legacy system.

5. Implementation

During the implementation, we experienced that it was comparatively easy to write the Java/CORBA code used to implement our design in spite of the fact that our knowledge of these technologies was limited.

Usually you do not have to think of a Java object being a CORBA object or not. But when parsing objects between CORBA objects you must be aware that only CORBA defined types (either standard types or IDL defined types) can be used as parameters to CORBA objects. A plain native Java object cannot be used. A JDBC database connection is a native Java defined class, and as such, it cannot be used as a parameter to a CORBA object. This caused a problem because we wanted several CORBA objects to use the same database connection, and thus needed to transfer the connection as a parameter.

Another issue to take into account, is that CORBA objects may reside in different processes. This is important to notice when objects contain non-data attributes - like an open socket of file descriptor. So even if we could make an IDL description of the database connection, it could not be passed across a process boundary, because it contains a connection to the database.

The solution was to introduce a single class by collecting all methods requiring a database connection for querying and manipulating the database. By designing systems with this kind of characteristics, we recommend to consider the connections to external resources.

5.1 Experiences with Oracle Application Server

Even though, we knew little about CORBA it was very easy to add CORBA functionality to our Java classes. This was due to the fact that Oracle has hidden the CORBA layer within the Application Server. When a Java class should be implemented as a CORBA class we needed to write the interface specification for this class. The interface is a Java interface that extends a special JCORremote interface that Oracle has defined. Then all we had to do was to deploy our Java class and interface specification to the server in order to make it a fully functional CORBA class.

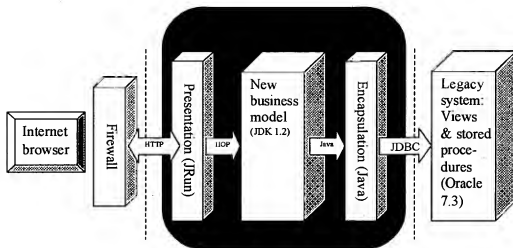
However the platform caused us several problems.

- In order to implement the Singleton pattern [5] for the manager classes we needed methods for storing and retrieving object references. A name service will usually be used to implement the methods but OAS does not support CORBA Name Service properly to be used for this. Other implementation techniques also failed and we were not able to find a solution that worked in OAS though it was crucial for the implementation.
- OAS has its own approach in support for CORBA functionality. This means you cannot use a lot of the existing literature and examples on Java and CORBA like in [6]. You have to use the CORBA documentation from Oracle and we found several situations where it wasn't sufficient.
- During the deployment process of Java classes to CORBA classes on the server, we experienced that OAS failed to deploy the classes when the number of classes exceeded a certain number. The reason seems to be a memory error, and we had to split our Java application into several applications on the Application Server in order to solve the problem.
- The automatic IDL generation in OAS reports very little on errors that might occur during compilation. This makes it very difficult to detect compilation fails.
- It takes a comparatively long time to deploy even a "small" application in OAS, which makes the edit-compile-test cycle a very heavy process for the developers.
- Finally we experienced a bootstrap problem, because some CORBA helper classes are generated on the time of deployment. The problem is that you need the helper classes when you compile your Java code i.e. before deployment.

The problem in implementing the manager classes as Singleton classes prevented us from getting the system to run with OAS. Though a solution probably exists, we were disappointed not to be able to find it within the project period. Thus we were forced to switch the implementation platform in order to get the system running.

5.2 Switching the ORB

Because of the problems experienced with OAS we decided to switch the implementation platform. The new platform was based on public available tools that we could download from the Internet and is illustrated below:



We decided to use the built-in ORB from JDK 1.2 [7] to implement the business classes. We also decided to implement the presentation layer as plain Java servlets, which are Java classes that dynamically generate HTML pages for a web server. We selected JRun 2.3 from Live Software Inc. [8] to handle the servlets. JRun is a Java servlet (and JavaServer Pages) engine. It also includes a built-in Web Server that we used to test our application.

Moving the developed code from OAS to the new implementation platform went smoothly though it required code changes, primarily eliminating OAS specific details. The changes made the code more conformant to traditional CORBA/Java development and involved the following:

- We had to write all the IDL ourselves and generate it to obtain the Java classes. We used a trick to catch the temporary IDL files while OAS was deploying the application. These IDL files were then edited to exclude OAS details, and then generated for Java. In other projects the IDL design is an important activity and our OAS trick did not give the most positive solution, but the solution worked!
- The creation of CORBA objects was different. The new code was more simple and should only make a new Java object instance, and connect it to the ORB. We were pleased to find that these changes were isolated to the factories because we used the Factory Method pattern (see section 4.3).
- The Singleton pattern was implemented using the CORBA Naming Service.
- The presentation layer was completely rewritten to Java servlets. This may sound like a major task but the calls to methods in business classes and the generated HTML code were the same and could thus be plugged into the servlet code.
- Minor changes were needed like extending the implementation classes from the generated base classes.

In general these changes were caused by our initial choice of OAS. We believe fewer changes are required to switch between other ORB implementations that conform better to the CORBA standard.

One benefit from an application server like OAS is that it includes facilities to set up multiple applications and configure how they shall run in different processes. It is designed to handle large applications and optimise performance for the created objects. No equivalent tool is delivered along with JDK and we had to implement this ourselves. A simple server program was developed so that all objects were created within this server process.

We found that debugging was a lot easier on the new platform because we could see the output from the server process.

6. Conclusion of experiences

We have introduced a good method to extend legacy systems with new functionality. The use of object-orientation made it relatively easy to build an encapsulation layer on top of the legacy system. We believe the principles can be used in general.

Eventhough, we had little knowledge of this technology we benefited of being mentored by an external resource. We also had success with the use of patterns.

CORBA and Java give a fine platform to implement the architecture we decided to use. CORBA is almost hidden to the client and partly for the server application too. However, in the implementation design you must be careful only to pass CORBA defined data types in calls to CORBA objects. A consequence seems to be that the design must introduce one monolithic class with methods for all the functions requiring a database connection because a JDBC connection is a native Java defined class. We believe our observation is general for similar applications based on Java and CORBA.

We were disappointed with our experiences with the CORBA support in the current version of Oracle Application Server. The Singleton problem actually prevented us from implementing our design on the platform. A key problem is that the product tries to hide CORBA too much for the developer. This makes it difficult to use traditional CORBA techniques and existing literature. We strongly believe that it is better to use tools that really conform to the standard.

We were pleased to experience that switching the ORB platform required very few changes in the code. Having spent more than 600 hours trying to get the system running on OAS it only took two persons one day to switch to the ORB in JDK1.2. It really proves the benefits from using standards like CORBA!

Performance considerations had low focus in our project. The use of the new system will not be very intensive and thus performance is not very critical. We expect the implementation will perform adequately but in a mission-critical system you will probably have to spend extra time checking for unnecessary communication overhead between different objects.

7. References

- [1] Centre for Object Technology, www.cit.dk/COT.
- [2] RAMBØLL Information Technology, www.ramboll-it.com
- [3] Allan. R. Lassen. *WorkSAFE/IKIS empiri*. COT/4-09-V1.0. (Internal working paper in Danish).
- [4] Johnny Olsson, Lisbeth Bergholt, Aino Cornils. *Mønstre - en indføring i analyse-, design- og arkitekturmonstre*. COT/4-07-V2.1. (In Danish).
- [5] E. Gamma, R. Helm, R. Johnson & J. Vlissides. *Design patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1994.
- [6] R. Orfali, D. Harkey. *Client/Server Programming with JAVA and CORBA*. 2nd edition. Wiley, N.Y., 1998.
- [7] Java Development Kit Software, <http://java.sun.com/products>
- [8] Live Software Inc., www.livesoftware.com

Transparent Caching of Web Services for Mobile Devices

Kamal Elbashir

Multi-Agent Distributed Mobile and
Ubiquitous Computing Laboratory
Computer Science Department
University of Saskatchewan
(306) 966-4744

kae501@cs.usask.ca

ABSTRACT

A Web service is a collection of functions packaged, published and consumed using standard Internet protocols. This paper presents a generally applicable architecture for transparent Web Service caching, with a focus on Mobile Devices. This approach defines a set of required Semantics embodied in a document called the 'Service Semantics Description' (SSD). The SSD is used by the Client-side cache and (optionally) by the intermediate Caching Proxy. A tool facilitating definition of semantic tags is developed as an IDE extension. Preliminary results are presented outlining the feasibility, transparency, general applicability, and initial measurements of CPU and Memory overheads.

1. INTRODUCTION

Consuming a Web Service is accomplished by sending service requests (method calls), the service responds with a method return. Requests and responses are couriered as SOAP (Figure 1) messages (Simple Object Access Protocol). Accompanying a Web Service is the service's WSDL (Web Service Description Language), specifying syntax metadata. An optional layer, the Universal Description and Discovery Interface (UDDI) offers a directory service for discovery of web services [3, 4, 7, 12].

Communication with a Web Service is established using standard Internet protocols (e.g. HTTP). HTTP is the protocol underlying the World Wide Web. Adoption of Web Services gains remarkably from the utilization of well established protocols of the heterogeneous network of the WWW [4, 12].

A Web Service client must obtain the service's WSDL document. The WSDL document contains syntactic information about the service (Namespaces, Method signatures, Data types and a URI). A client consumes a web service by initiating the exchange of SOAP messages (see Figure 1-A). The messages must follow the inscriptions defined in the WSDL document (URI, Signatures, and Data Type information) [4, 12].

For a mobile device, consuming a web service is an attractive interoperable approach to access remote business logic. Mobile devices are constrained by Battery, CPU, Memory and Weak Connectivity. Weak Connectivity results from intermittent communication due to low bandwidth, high latency or expensive networks. Furthermore, Mobile devices are susceptible to both voluntary and involuntary disconnections (user initiated disconnections vs. disconnections due to a change in the availability of a resource, e.g. Battery life). For the mobile user, constraining productivity to times of full connectivity hinders a successful and usable deployment of remote business logic [1, 7].

This paper will refer to a mobile device consuming a Web Service (a Web Service Client) as a Mobile Host (MH). A Cache is a temporal memory coordinated by a Caching Policy, cache records are memorized instances of data (Requests/Responses). A Proxy is an entity acting as an intermediate connectivity tunnel, a proxy may be caching (Caching Proxy) [6]. A caching proxy plays an intelligent role while tunnelling requests to their respective endpoints. Cacheability of a request (method call/return cacheability) is the property stating that the request may be cached while maintaining application logic (no negative side effects on the application logic) [3, 8].

2. PROBLEM DEFINITION

2.1 Introduction

Operation of a mobile service client is restricted to times of connectivity. The service Provider may suffer network or system downtimes. Furthermore, the mobile device is constrained by Weak network connectivity. When the mobile client is disconnected (null connectivity) the service is inaccessible, and the mobile application is unusable. Null connectivity occurs when the device is out of network range, or when the device is voluntarily or involuntarily disconnected (e.g. user-initiated power-off vs. a depleted battery state). Additionally, network infrastructures supporting mobility are bandwidth-limited. Clients consuming rich services suffer degradation of response times due to latencies incurred when sending large upstream requests (service arguments) and receiving of large responses (return values).

This paper will explore the issues present when a MH loses network connectivity (null connectivity) and the necessary recovery logic upon reconnection (reintegration phase). The user experience of a mobile device is reflected by the user's inability to continue working while the MH is disconnected. An improvement of the user experience can be achieved by enabling the user to continue to work while in disconnected mode. Increased application response times, due to caching of frequent requests and/or Prefetching of anticipated future requests is a desirable improvement.

The mobile application is assumed to be resilient to stale resource access (invalid, out of age resources). This implies that the application logic is not broken when a request is answered from the cache. This is a necessary assumption since the consistency of a cached resource cannot be guaranteed while operating in null connectivity.

2.1.1 Web Service Caching

A Web Service, described by the WSDL document can be viewed as a Remote Object, accessible by SOAP messaging over HTTP. Methods and properties exposed by the remote object are accessed using SOAP messages adhering to the calling convention prescribed by the WSDL. Every SOAP request/response corresponds to a single method invocation or a single property set/get operation [3, 4, 7, 12].

A naive caching architecture stores tuples of request/response pairs. When a new request is made; and a matching request is in the cache, then the corresponding response is returned (cache hit). If a similar request is not in the cache (a cache-miss); and the MH is operating in disconnected mode; then an exception is raised. All attempted requests, while in disconnected mode, are stored in a 'Pending' FIFO queue for execution when connectivity is restored (Reintegration Phase). Upon reconnection, the reintegration phase commences by issuing pending requests, an attempt of synchronizing the state of the disconnected cache with the state of the remote object.

The relationship between method calls and the state of the service is crucial. Service methods fall into one of three types: Read-methods (state-reading) and Write-methods (state-altering), or State-independent methods. Properties offered by the service are inherently state-reading and state-altering [3, 8]. The following discussion is only concerned with service methods; the outcome can be generalized to service Properties.

The classification of a method as state-reading, state-altering, or state-independent is best known by the publisher of the Web Service, since services implementations are exposed as black-boxes. It is impractical to assume that all methods are of one type or another (in reference to their state dependency). There is no standard for specifying Semantic information about Web Service methods. The WSDL document is limited to describing only the Syntactic information relating to service consumption [3, 8, 12].

A Web Service caching architecture, built specifically for Mobile devices must consider the limited processing and space constraints available to the mobile device (MH). These constraints limit the allowable Cache-size, and the processing requirements of cache management and operation [1, 3]. The architecture should preserve the logic consistency of the mobile application, a mobile application functioning with and without a cache should experience no side effects resulting in incorrect operation. Out of age cache records should be invalidated. Cross-MH cache consistency is a requirement for mission critical mobile applications (e.g. shop floor applications) [1, 3, 8].

Caching in the area of the World Wide Web is focussed on Read-caching [5, 6]. Read/Write caching has been well explored in the areas of File systems, Databases and Distributed Object Systems [2]. Many issues faced by a mobile Web Service client are comparable to aspects of caching in the aforementioned paradigms. This paper explores previous research in section 3.

Approaches common to Web caching are severely limited when applied to caching of Web Services. Static web pages need only Read-caching. Only when a sophisticated web caching system is utilized (e.g. Active Caching, for dynamic web content) then a form of Write-caching is present [5, 6]. A modified web caching proxy may treat the Web Service as a dynamic page and cache the SOAP messages that are courtered in the body of HTTP requests. Such architecture is very limiting and its assumptions are unrealistic. The modified caching proxy must compare bodies of

SOAP messages when testing for a cache-hit, the message body may include metadata that is not necessarily part of the method call (e.g. RequestID), resulting in false cache-misses [3]. Another problem with such an implementation is the proxy's inability to replay state-altering requests upon restoration of service availability. Finally and most importantly, the meaning and structure of a web page is fundamentally different from that of a web service, the required semantic information enabling caching is hardly translatable into the domain of web caching.

2.1.2 Mobility and User Experience

A cache for mobile devices must support transparent switching between connected mode (remote-execution) and null connectivity mode (cache-based execution).

Furthermore, service methods requiring large arguments incur delays due to network transfer latencies. Similar latencies are incurred when methods return large dataset are executed. Repeated duplicate requests should only be returned from cache and bandwidth utilization should be minimized when possible [7, 8].

2.1.3 Caching-Architecture Deployment and Existing Web Services

A caching architecture may require the deployment of specialized components on the server-side, a difficult requirement in real-world scenarios, service Providers are generally reluctant to alter existing implementations. An approach to transparency is achievable by caching independently of the service implementation, utilizing a Caching Proxy [7, 8]. Caching schemes employing an intermediate proxy are extensively used for caching in the areas of the WWW and Distributed Object Systems. An intermediate proxy implementation appears to be the service client when viewed by the service provider. When viewed from the client's perspective, the proxy appears as the original Web Service.

2.1.4 Cache Location

A client-side cache (see figure 1-B) permits the client application to recover from service requests when the service is unavailable (network outage or service downtime). If the client is caching independently of the service, unnecessary executions on the server if the client issues an expensive request and network connectivity is lost before a service response is received. Furthermore, notifications of invalidated client-cache records are harder to implement since the server is unaware of the client cache. Finally, multiple caching clients are not centrally coordinated, the clients maybe within range to form an Ad-hoc network and exchange cache-hits while the service is unavailable.

A second cache (see figure 1-C), residing at an intermediate layer between the mobile device and the web service is a promising approach. The intermediate cache (caching proxy) must not be susceptible to network disconnections (see Figure 1). Such an organization permits proxy-led coordination of client caches. The service is now shielded from processing duplicate requests made by multiple mobile devices. A proprietary protocol, optimizing the link between the intermediary proxy and the mobile device is now possible (e.g. offering compressed streams). Invalidation reports are more readily deliverable to mobile clients, as the intermediate caching proxy becomes the centralized cache coordination layer.

2.2 Key Questions

From the previous discussion, several questions arise:

1. What metadata is necessary to enable caching of Web Services? How can I enable a developer to specify the necessary metadata with minimal effort?
2. What should be cached?
3. How can a cache that is independent of service implementations be implemented?
4. What consistency guarantees are attainable?
5. What prompts cache invalidation? What happens in the reintegration phase?
6. How does existing research help in designing a practical caching architecture for Web Services?

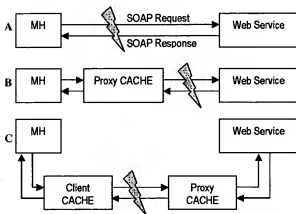


Figure 1: MH - Web Service Connectivity without a Cache

3. RELATED WORK

3.1 Web Caching: A Survey of Web Caching Schemes for the Internet

Wang's paper [5] is a survey of caching architectures on the Web. Proxy servers are recognized as an effective solution for bandwidth optimization, availability, and scalability. Web Proxy servers sit between the web client and the web server, mimicking the behavior of the real server. Caching may occur on the server, the proxy or the client. A set of significant improvements are recognized as following [5]:

1. Web caching optimizes bandwidth usage by reducing network traffic as more and more resources are found in the cache.
2. Web caching reduces access latency due to the fact that documents may be found in the local cache (on the client) or on nearby proxy servers, reducing required network traffic when fetching documents. Because of the traffic reduction, more bandwidth is available for a cache-miss to be retrieved quickly.
3. The web server's performance is improved as more requests do not reach the server and documents are retrieved from the client or proxy caches.

4. Improved robustness, document availability is significantly improved due to replicas existing in local and remote caches. If the server goes offline, cached documents are still available to interested clients.

5. Proxy servers act as intermediary traffic deltas, data collection and statistical analysis of access patterns is more easily doable.

6. Load balancing is in effect, as proxy servers reduce the server's utilization by forwarding requests to the least utilized server.

Wang recognizes disadvantages of web caching, most importantly is cache consistency maintenance, as stale records may be served to clients while the server is offline. Increased request overhead, due to processing by intermediary proxy (or proxies). A proxy is also recognized as a point of failure, the proxy's availability becomes more critical than the web server's availability. This is because one proxy server may act as a caching agent (delta) for multiple web servers [5].

Wang outlines ten desirable properties of a web caching architecture, namely: Fast Access, Robustness, Transparency, Scalability, Efficiency, Adaptivity, Stability, Load Balancing, Ability to deal with Heterogeneity and Simplicity. A number of caching architectures are analyzed including Hierarchical, Distributed and Hybrid. Hierarchical caching producing the shortest connection latencies when compared to distributed caching. On the other hand, distributed caching is found to have the shortest retrieval latency but with greater bandwidth utilization [5].

Two common approaches to cache routing and optimization are recognized, by growing a caching distribution tree formed by nodes of intermediary proxy servers. Cache resolution is performed by a routing table or a hash function [5].

Prefetching [9] is needed in order to maximize the cache-hit rate, this is done by anticipating future requests and prefetching the corresponding documents pre-demand. Prefetching may be executed between the client and the web server, between the client and the proxy agent, or between the proxy agent and the web server. Caching between the client and the web server is recognized for increasing the cache hit-rate by 45% at the cost of doubling network traffic. Rate-controlled prefetching is recognized as a possible solution to the unacceptable increase of network traffic [5].

Prefetching between the proxy agent and the web server provides significant improvements over the previous approach offering a very good predictability of client access patterns at the proxy [5]. Furthermore, due to the decreased network traffic downstream to the client. Prefetching between clients and proxy agents may best support caching for mobility, as the client and the proxy agent cooperate to cache and prefetch documents without the server's intervention [5].

A number of cache placement and replacement strategies are identified and analyzed in regards to cache size/speed. A good cache placement/replacement strategy is very important when caching for mobility. Two types of cache coherencies are recognized, Strong and Weak. Strong cache coherency is maintained either by client validation or Server invalidation. The former results in higher cache-hit latencies, while the latter requires server cooperation. Server cooperation may be needed for broadcasting invalidation reports. Weak coherency is the case when cache records are invalidated by a timeout (TTL value). Piggyback invalidation requires the server cooperation, by

sending lists of invalidated items attached to responses of new requests. The requirement for cache coherency depends largely on the client's tolerance for stale resources and on the frequency of resource changes [5].

Proxy placement is very critical for optimal performance gains. The desired properties inherit many of the desired properties of a web caching system [5].

Caching of Dynamic web resources, a topic of interest when considering caching of web services is briefly touched upon. Read-caching is the focus of the majority of web caching strategies and studies [5]. The two widely used approaches to dynamic-resource caching are identified as Active Caching and Server Accelerators. The former requires computation applets attached to dynamic portions of a page to be sent to the proxy and the latter is done by exposing a set of caching APIs at the accelerator entity, the dynamic resources at the server must utilize the APIs in order to achieve caching of dynamic resources [5]. Both approaches violate at least two of the desired properties outlined at the beginning of the paper, namely the ability to deal with heterogeneity and simplicity.

3.2 Web Service Caching for Mobility

Terry et al. [3] discuss the need for caching of Web Services to support mobility. The motivation is to offer disconnected operation of web services. Transparent deployability and General applicability are two desired properties of a web service cache. The paper distinguishes attributes specific to caching of web services, when compared to caching in the areas of file systems, and databases as both offering standard interfaces with well known semantics (Read/Write and Query/Insert/Update/Delete), and Web caching is limited to read caching [3].

The authors use the .NET MyServices set of services for an experiment in caching for disconnected operation. .NET MyServices are services offering personal profile maintenance and a contacts directory. The exposed operations are Query, Insert, Update and Delete operations [3].

The proposed architecture utilizes an intermediate SOAP proxy agent, caching SOAP requests/responses made by service clients and serving cached responses when the client is in disconnected mode. Cached requests are queued up for replay (playback) at the reintegration phase [3].

Two critical issues surrounding caching of Web Services are identified, namely Cacheability/Playback, and cache Consistency maintenance. For an effective web service cache, the authors identify the requirement that all service operations must be designable as Update or Query operations (Read/Write semantics). The lack of semantic information in the service description (WSDL) creates a challenge for caching consumers of boxed web services. Query operations are deemed cacheable if they do not alter the server state (e.g. server logs). Update operations are operations that are state altering on the server [3].

Maintaining strong cache consistency on a mobile client is deemed unachievable by the inherent property of weak connectivity. Consistency is also explored when execution of an operation invalidates a stored response of another. A proposed solution is to invalidate the old record, or apply a modification transformation for the in-cache record. The authors declare that "for preexisting Web services, understanding the correct consistency requirements is an extremely challenging issue" [3].

The effect of a web service cache on the User experience is identified as a crucial criterion when evaluating an effective web service cache. Ideally, the user should not be aware of disconnections but this is identified as a difficult goal. An additionally challenge is the ratio of consistency guarantees offered by the cache and the quality of the user experience. Altering the client to be cache-aware, and to display hints to the user regarding the active consistency guarantees may have a positive effect on the user experience [3]. The later proposition does not satisfy the property of transparent deployability outlined at the beginning of the paper.

Terry et al. discuss the effect of differing structural formatting of SOAP messages on a web service cache. This is identified as a possible problem when comparing requests for similarity. WSDL is identified as providing enough information for an intermediate proxy to fabricate a fake response to a service request. Although the lack of a specification mechanism for Default values may limit the range of possible fabricated responses [3].

Prefetching (hoarding) is identified as a mechanism to maximize cache-hit rates. An implementation would employ an algorithm for anticipating requests for prefetching. The lack of semantic information about the service and the lack of a standard mechanism for users to specify a set of requests for hoarding complicate a cache implementation supporting prefetching [3].

Finally, maintaining application Security is outlined as an important consideration. Though is complicated by lack of standards regarding authorization of access to a web service operation. An intermediary Proxy, caching for multiple clients, may open a set of privacy/security holes, this is relevant when cached responses differ by the authenticated user [3].

3.3 Caching of Objects in Distributed Object Middleware (CORBA) for Mobility: Dominant

Distributed object middlewares offer remote method execution, platform interoperability, and location-transparency of objects. The first two properties offer the closest resemblance of the Web Services paradigm.

Conan et al. [2] describes an architecture offering disconnected operation of a CORBA environment for mobile clients. Dominant, uses portable interceptors (PI), a CORBA mechanism for peaking into and altering of the communication between a client and an ORB (Object Request Broker). Dominant offers continued operation in partial and null-connection modes with minimal or no overhead when operating in connected mode.

The transparency of utilizing CORBA's portable interceptors enables connection awareness to be shifted away from both the client and server objects and into the Dominant middleware extension. Dominant works by intercepting requests made to the CORBA ORB and transparently rerouting requests to a local disconnected object [2].

Several performance protective measures are employed. In order "not to punish strongly-connected clients", while strongly connected, client's requests go directly to the remote object [2]. Also a hysteresis mechanism is proposed for handling variations in connection availability. An interface to the hysteresis mechanism maybe consumed by the client application in order to alert the user to changes in the connectivity-mode, also offering the user the ability to voluntarily disconnect. Three connectivity modes are recognized, namely: disconnected, partially connected, and connected. Transparent switching between modes is activated

at the time of a client request. A set of inputs is required when deciding on an operation to execute, the inputs are: disconnection mode (voluntary or involuntary), the mode of the last request (to the same object), the operation name, and the network/object current connection mode. A matrix is developed allowing correct state transfers in various modes [2].

In the connected mode, the requests are immediately sent to the remote object. In the partially connected mode; the operation is executed both locally and remotely, depending on the call semantics (presence of in, out, in/out parameters and if a return value is expected). In disconnected mode, operations are executed locally, and are logged depending on the semantic relationships with other operations. The log is vital at the reintegration phase and reconciliation may need to occur to maintain coherency between the disconnected object (the proxy) and the remote object. However, Domint assumes that no object is accessed by more than one disconnected client [2].

Preliminary performance evaluations, performed on a Windows CE device, show overhead of 14% to 1% when connected, 50% to 6% when partially connected, and from 20% to 6% when disconnected. The incurred computation cost is justified by the introduction of transparent connectivity, without modification to either the server or the client implementations [2].

3.4 Delayed Execution/Call Aggregation: Reducing Overhead of .NET Remoting

In the context of Web Services, .NET Remoting is the infrastructure implicitly in-use when .NET applications publish or consume Web Services. Remoting abstracts remote objects to behave as local objects. The Remoting infrastructure offers various extensibility options, the lowest level communication channel maybe replaced or customized, the messages to be exchanged maybe modified before or after formatting, and calls to remote objects maybe intercepted immediately after a consumer issues a request and before the request is propagated downwards in the remoting stacks. The latter is the mechanism commonly used in distributed object middlewares. The client accesses the remote object via a local proxy, known as the Transparent Proxy in .NET Remoting. The transparent proxy is generated at runtime by the Real Proxy (also a client-side object). The real proxy is generated when remote objects are referenced, and its binary maybe replaced or modified without modification to the object consumer.

Clegg [11] discusses the overhead introduced when employing .NET Remoting for remote execution. Clegg describes and evaluates an architecture that transparently monitors and optimizes calls to remote objects. RROpt is modeled on the DESORMI framework (Delayed-Evaluation, Self-Optimizing Remote Method Invocation by Kelly, Field, Bennett, and Yeung). The implementation is a modification to remoting-relevant code in the .NET CLI, namely the Mono CLI. Such an implementation eliminates the need for server/client modifications [11].

PROpt works by checking at runtime for candidate delayed calls, specifically by looking for methods of objects inheriting from MarshalByRefObject. When a remote call is incurred, it is stored in a delayed-list and a dummy return is pushed to the stack. If a method attempts to use the return value then the delayed method is immediately executed. A set of delayed methods is executed by formulating a plan encapsulating their data dependencies and forwarding the plan to the server. PROpt assumes that all servers are PROpt-enabled (executing on top of Mono CLI with PROpt

extensions). Argument aggregation is also performed when a set of methods share an argument [11].

Another optimization employed by PROpt is Plan caching, sets of previously executed aggregated-calls (a plan) are remembered on the server, a client refers to them by ID, furthermore decreasing network traffic requirements [11].

The remoting infrastructure protects applications by containing them in "Application Domains" (light weight processes). Multiple client accessing the same remote object are not aware of each other, multiple application domains may be hosted within one process. Furthermore, multiple remote objects on a single remote server maybe accessed by a set of application domains in a process. In order to aggregate cross-object, PROpt implements its aggregation targets per server name [11].

The speedup possible by PROpt did not prove to be consistent. PROpt performed well when data dependencies between methods existed. Outbound Network traffic is significantly decreased due to call/parameter aggregation. PROpt optimizations failed to materialize when no data dependencies exist between method calls, this is the case when the network infrastructure is fast [11]. No applicability to mobile clients is considered.

4. A WEB SERVICE CACHE

4.1 General Considerations

An architecture supporting a predefined set of services and a special client implementation is an application-specific cache. An application-specific cache is optimized for the application logic, all caching decisions (such as placement, replacement, prefetching) target optimal consistency and performance of the application. Proprietary communication protocols (such as ones supporting compression, or multicast notification of cache invalidation) are expected, as the application permits [3, 8].

A General caching architecture, on the other hand, offers a cache to any Web service. Such architecture faces many challenges, most importantly is the decision of cacheability of a web service request. A Web service is treated as a black box, requiring the availability of cache-hints (metadata) supporting decisions such as cacheability, invalidation conditions, and default responses (return values) [3].

A caching proxy is necessary in a general cache, in order to support independent caching decisions (independent of the client and the server). On the other hand, an application-specific implementation maybe embedded in the web service and client implementations.

When the goal of a cache is to improve the availability of a web service, then a larger Cache-size and optimal placement and replacement strategies are a priority. The existence of stale-resources (invalid cache records) in the cache is permitted, hence to improve the service's availability. Knowledge of the MH connection-state is necessary, in order to seamlessly resume returning of cached responses when the MH enters null connectivity. Cached responses are returned only when the MH enters null connectivity, in order to achieve the best-possible cache consistency. Newer requests overwrite their older counterparts in the cache.

On the other hand, when the goal of caching is to improve the application's performance, then cached responses can be returned even when the MH is in full connectivity mode. Such an approach may result in substantial response-time improvements, especially

for expensive web service methods: methods requiring a relatively large set of arguments, methods requiring an expensive or lengthy computation, and methods returning a sizable data object. Web service requests maybe aggregated to improve bandwidth utilization while returning cached responses. The downside to such performance optimizations is decreased cache consistency in relation to the real web service. A workaround to further improve the consistency of cached responses maybe achieved by periodically prefetching, or periodically submitting invalidation-queries of expensive cached responses.

4.2 Cache Location

A server-side cache offloads the server from re-computation of frequent requests. Such a cache implementation is commonly a specialized architecture. Cache-consistency is best obtained when using this approach, since invalidation reports maybe requested or broadcasted to the known caching proxy (or proxies). Another improvement is the transparency of the cache for a MH consuming the web service. A slight improvement in service availability is present due to a protection from server downtimes, as the server-side cache continues operation while the server is down. On the other hand, a MH suffering local null connectivity is also disconnected from the server-side cache.

A client-side cache offers the service's availability to the MH while in null connectivity [10]. Cache-consistency is minimally maintained in this approach. Near-time cache invalidation is harder to achieve since the MH cache is independent of the service implementation [3, 8].

An intermediate caching proxy offers transparent service caching for a MH. A caching proxy is more capable of tracking the list of caching MHs as it act as an intermediate delta between multiple MHs and multiple service providers. A shared cache is in effect, as requests from multiple MHs are cached for other MHs. An intermediate cache is best equipped, independently of the service provider, to deliver invalidation reports to the tracked list of MHs. The service remains unavailable to a MH in null connectivity, as the intermediate cache becomes disconnected.

A client-side cache assisted by a caching proxy is best equipped when the goal is protect the client from service unavailability while maintaining best-possible cache consistency. Several performance improvements are now possible because the client-side cache and the intermediate caching proxy can agree on a proprietary communication protocol supporting better request aggregation and near-time broadcasts of cache invalidation reports.

4.3 Semantic Metadata

The lack of semantic metadata f a web service methods presents a challenge for caching independently of the service implementation [3, 8]. The metadata may accompany the service as an extension to the service's WSDL document, or maybe maintained by a third party maintaining a repository of metadata records targeting a growing set of web services. An alternative is to allow the service consumer to specify an updatable set of meta tags, assisting caching decisions when determining cacheability and invalidation conditions. Client maintained metadata are specified per web service.

A web service method should be tagged if it is cacheable, and a default return value should be specified. The former aids the cacheability decision of the cache, while the latter offers a protection against cache-missed of a MH in null connectivity.

Additional tags may outline invalidation conditions based on time-values, age-thresholds. Method interdependencies will aid request-aggregation logic and can also provide further improvement when maintaining cache consistency by invalidating cached responses when state-modifying requests are executed.

4.4 Caching Policy

The request signature and argument values must be considered in hash function supporting cache placement of web service responses. Since multiple requests to the same method may differ on a single argument, while unique responses are always returned.

LRU, LFU and Size are competing replacement strategies when a decision relates to limiting the cache size, especially for a MH with space and computation constraints [7]. It is to be determined if any or a combination of the well studied replacement strategies are best suited for a web service cache supporting mobile devices.

A cached response maybe invalidated by its age or a timeout value. Furthermore, a cached resource maybe invalidated and evicted from the cache because an invalidating request was submitted.

Cache invalidation reports maybe broadcasted by the server or an intermediate proxy, or maybe piggybacked on the results of new requests. Furthermore, invalidation query maybe submitted on intervals or piggybacked on new requests. Limitations are present depending on the cache location and the number of entities supporting caching between the MH and the service provider [3, 8, 10].

Default return values are useful on a cold-start or when a cache-miss occurs while the MH is in null connectivity mode.

An alternative for recovery from a cache-miss when the original service is down while the MH is fully connected is by rerouting of requests to a service replica. This approach further protects the MH from service unavailability due to provider downtimes or peak hour unavailability.

The consistency of responses between the service replica and the original provider is an issue out of context for this research.

4.5 Prefetching/Hoarding

On a cold start, a MH may issue a set of predefined requests for caching. Alternatively, the service provider or an intermediate cache maybe store an up-to-date image of most frequently requests methods and push them to the client on a cold start.

Prefetching offers substantial improvements in response times to most frequently requested methods, at the expense of higher bandwidth utilization. The negative side effects of prefetching maybe overcome by request-aggregation and by adaptive prefetching logic with explicit awareness of network QoS [9].

4.6 Client Sessions

A MH utilizing a communication channel supporting session state can benefit from an intermediate cache retaining a MH-tailored list of frequent requests. Prefetch or hoarding requests can be initiated on the behalf of the MH.

For a frequently disconnecting MH, expensive requests made while in full connectivity maybe pushed upon reintegration. The minimum improvement is in effect when a cache-refresh cycle (or a cache-replacement function) is executed and an expensive request is not evicted because the owner MH connectivity is considered.

5. PROPOSED ARCHITECTURE

The design of the proposed architecture is modelled to support the following two scenarios and their consequences:

1. Null Connectivity: The MH enters null connectivity, on a new request the following conditions are evaluated:

A) Cache-hit, the cached response is returned. A state-altering request is queued into the Replay queue. State-reading requests are queued into a Delayed-fetch queue, a mechanism for improving cache consistency at the reintegration phase.

B) Cache-miss (Cold Start), a default return-value is returned. A state-altering request is queued in the Replay queue. A state-reading request is queued for delayed fetch.

2. Full Connectivity: The MH enters full connectivity, the following conditions are evaluated:

Non-empty Cache:

A) Cache Miss:

New requests go directly to the Web Service, and when a response is received, a request/response tuple is inserted into the cache.

B) Cache Hit: State-altering requests are sent directly to the service, a request/response tuple is cached. A response to a state-reading request is returned from cache (if the response is valid or if the request is long-living), the state-reading request is queued for delayed-fetch, a mechanism for improving cache consistency in the long-run. A long-living request is a request with a long cache TTL or the time (t) of the cached response is less than the invalidation time (On Time) of the method.

Empty Cache (Cold Start):

A) On start, if an intermediate proxy exist, request a cache Image, Done.

B) On start, if a Prefetch-list is known, queue all items from the list into the Delayed-fetch queue, Done.

C) On a new Request and a Cache Miss: Return the default return-value associated with the request and queue state-altering requests into the Replay queue. State-reading requests are queued for delayed-fetch.

For effective caching of the Web Service this approach uses a document encapsulating the Semantics of the Web Service. The encapsulated semantics are shareable, and are either defined by the service Consumer (at development time) or by the service Provider. This paper refers to the document encapsulating the service semantics as the Service Semantics Description Document (SSD Document), an XML document. The SSD also specifies Hints adding various cache operations (Invalidation Conditions, Prefetch Lists, and addresses of a Replica and Intermediate Proxy).

A developer tool that integrates within the IDE of Microsoft Visual Studio.NET is provided to aid a mobile application developer to seamlessly specify an SSD. A similar tool is also provided for a Web Service developer in order to specify an SSD. The developer tool allows transparent incorporation of a cached Web Service while decoupling the programmer from the caching infrastructure.

The Service Semantics Description document specifies the following metadata regarding each service Method:

1) Cacheability: specifying if a method should be cached or not. A method is non-cacheable if a cached value will always lead to faulty application logic (e.g. a GetLastRequest method).

2) Replay: upon reconnection, this tag hints if a method should be replayed or not. To maintain application logic, a State-altering method should be tagged for 'Replay'. A method may not be tagged for replay because of one of two reason; either the method is state-reading or the method's state-altering behaviour is irrelevant upon reconnection.

3) Default Return Values: the value of this tag is a serialized-graph of a meaningful default return value for a method. This tag enables the MH to partially recover from a Cache-Miss while in full connectivity, or when a MH cold starts. It is expected that only cacheable (state-reading) methods will have default return values. State-altering methods can not have default return values as this may lead to illogical returns (e.g. a Bool CreateRecord() method, returning success or failure of record creation).

4) Invalidation Conditions: a cached response should be invalidated after a specified Age (in minutes), or after a certain time of day, or when an invalidating hint is received (or fetched), the latter is not implemented.

5) Method Interdependencies: for N methods, this is an N x N table specifying interdependencies between service methods. Effective Request-aggregation can only be implemented if method interdependencies are known, the current prototype does not implement request-aggregation. A cached response is invalidated if a state-altering request, that is also a dependency of the cached response, is submitted. A table lookup is used to invalidate a cached response.

6) Prefetch Parameters: a list of method names and proper arguments is specified to enable prefetching on a cold start or at periodic intervals.

7) Intermediate Cache Proxy: a URI specifying the network path to an intermediate Caching Proxy, the caching proxy interface appears as a replica of the original Web Service.

8) Service Replica: a URI specifying the network path of service Replica. A Replica is utilized when a service downtime is detected.

The implementation of the Cache is built as a Proxy of the Web Service, the Proxy exposes an interface identical to the real service, decoupling the service consumer and service provider from the cache. The Proxy implementation is essentially a disconnected Object providing transparent access to the real service while continuously monitoring service availability and network connectivity of the MH. This object is referred to as the Caching Web Service Object (CWSO).

The CWSO implements a Hashtable for storing tuples of request/response pairs. A connectivity monitor detects service availability by requesting the service's WSDL at predefined intervals (5minutes). The connectivity monitor has OS hooks to detect network connectivity at the MH, entering the CWSO into one of two states: Full Connectivity and Null Connectivity. A Replay FIFO queue and a Delayed-fetch FIFO queue are used for

queuing state-altering requests (the former) and queuing of prefetch-requests (or delayed-fetch requests).

The SSD document is stored as an XML file accompanying the Real Proxy assembly. The CWSO provides an interface for consumers wishing to alter the service semantics or the default caching policy at runtime.

The Cache hashtable contains wrapped IMessage objects (CacheEntry). An IMessage object specifies the method's signature, argument list, call context, and method's response. The CacheEntry object includes the request time and time of the last cache-hit, along with the object's size in bytes.

The Cache Manager executes at state transitions (Full Connectivity and Null Connectivity) and appropriate action is taken. The Replay queue is processed before processing of the Delayed-fetch queue, to allow state changes to occur before processing of new state-reading operations. The processing of the Replay queue is started at a random interval between 1-5 minutes after achieving full connectivity. On a Cold-start, the replay-queue is empty and processing commences with the delayed-fetch queue instead.

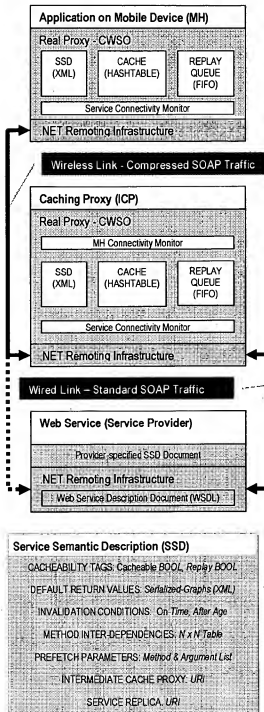
The Intermediate Caching Proxy (ICP) hosts an exact copy of the MH's CWSO, plus an additional connectivity monitor targeting MHs. Connectivity session management between the ICP CWSO and the MH's CWSO is planned as future work.

The link between the MH and the ICP is a wireless link susceptible to disconnection and low QoS (factors of weak connectivity), SOAP communication is tunnelled over a 'customizable' HTTP channel (e.g. Compressed). The link between the ICP and the real Web Service is assumed to be a wired link, offering higher bandwidth and strong connectivity, the communication is couriered by standard HTTP.

When the MH's CWSO is accessing either the real Web Service or the ICP, the logic is consistent. A replica Web Service appears to be the real Web Service to both the CWSO and the ICP. The MH or the ICP both appear as simple clients to either the Web Service or the Replica.

If a Service Connectivity Manager detects a service downtime (failure to return a WSDL document), requests are transparently routed to a Replica (if exist) or the Cache. If the connectivity manager detects null connectivity at the MH, then the ICP, the real Web Service and the Replica are all disconnected, and all requests are routed to the local Cache.

Cached responses are invalidated by Age or Time (from the SSD), the detection of an invalid CacheEntry happens when a cache-hit is suspected or when the CWSO executes the cache's SizeManager (every 20mins). CacheEntry-objects maybe evicted from the cache if the cache-size exceeds a threshold (predefined as 10mb), the eviction strategy maybe LRU or SIZE.



6. PRELIMINARY EVALUATION

6.1 Experimentation Plan

Two experiments have been performed, the first experiment measures the Overhead introduced by CWSO on the MH. The second experiment captures the State Transitions when controlling factors are toggled.

The scenarios are controlled by the following factors:

- Network Connectivity: Connected and Not Connected
- Service: Available and Unavailable
- Request: State-reading R, State-altering W
- Cache Test: Hit or Miss

The service provider in the Experiment 1 is on the same host as the emulated MH. This decision was made to eliminate network latencies from the experimental results. All performance data in this experiment is collected at the local MH.

The service provider in the Experiment 2 is a remote host. The collected performance data is local to the MH.

A mobile network was not utilized in these experiments, this should not skew the result since this implementation does not introduce additional communication.

The link between the ICP and MH's CWSO is a standard HTTP stream, SOAP Compression is not yet implemented.

Cooperative Caching and Prefetching has not been tested.

Network, Service, Replica and ICP availability is simulated by object parameters.

The active replacement policy is LRU.

6.2 Test Suit

6.2.1 Hypothetical Web Service (HWS)

The service exposes 4 methods, R denotes a State-reading method, W denotes a State-altering (write) method:

OUT R_1() consistently returns a constant value.

OUT R_2(in) is a function of in.

OUT W_1(in) is state-altering returning success/failure.

VOID W_2(in) is state-altering without return.

The 'in' argument to method R_2 is an Integer, W_1 and W_2 'in' arguments are of type String, the String arguments vary randomly in size between 100bytes and 1kb.

The return value of R_1 is a String, R_2 is an Integer and W_1 is Boolean.

The associated SSD is available in [13].

The goals of this experiment is to test processing overhead (CPU) and cache-size overhead (Memory) at the MH's CWSO when 1000 requests are sequentially executed.

To calculate CPU and Memory overheads, the experiment is run twice, for each method. The first run is performed without the CWSO, the second run is with the CWSO. CWSO initialization times are accounted for.

6.2.2 I-Help Web Services

I-Help is a real-world public discussion forum system, utilized mainly by Computer Science students at our department. Actual User Traces were not collected for this experiment, instead the

goal of this experiment is to verify the architecture's general applicability to existing Web Services and secondly to verify the system's State Transitions when controlling factors are toggled.

I-Help Web Services exposes two operations of interest, a Query operation and a Post operation, the associated SSD document is available in [13].

6.3 Experimental Conditions

Mobile Application: Proof of Concept Client

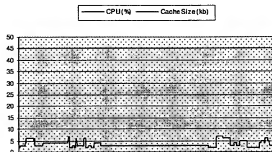
MH: Emulator Windows CE.NET 4.2.

Experiment 1 Service Provider: .NET Assembly

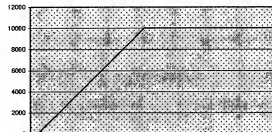
Experiment 2 Service Provider, 3rd-party implementation: Axis, Java Web Services

6.4 Preliminary Results

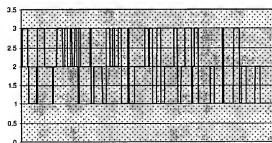
6.4.1 Experiment 1: Overhead



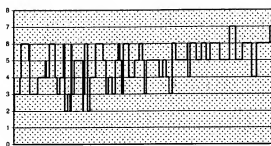
CPU and Cache-Size, Experiment 1, Method: R_1:



CPU and Cache-Size, Experiment 1, Method: R_2



CPU and Cache-Size, Experiment 1, Method: W_1



CPU and Cache-Size, Experiment 1, Method: W_2

6.4.2 Experiment 2: State Transitions

Network Connectivity	WS Availability	Cache Hit	Operation	Execute at Service	Cache	Default	Delay Fetch	Reply
Y	Y	Y	R		1		1	
Y	Y	Y	W	1				
Y	Y	N	R	1				
Y	N	N	W			1		1
Y	N	Y	R		1		1	
Y	N	Y	W		1			1
N	Y	N	R			1	1	
N	Y	N	W			1		1
N	Y	Y	R		1		1	
N	N	Y	W		1			1
N	N	N	R			1	1	
N	N	N	W			1		1

6.5 Summary of Results

The results show that the CPU overhead at the MH, expended in the Real Proxy object by the components: Service Connectivity Monitor, Cache-Hit Test, Cache-Size manager and the Queues, did not rise above 10%, as a preliminary result this is acceptable.

The SSD's description of W_1 as non-cacheable successfully resulted in a Cache-size of 0bytes. W_2, marked with a VOID return resulted in a similar outcome. R_1 returning a constant occupied 45kb in the Cache, a value indicative of the size of initial CacheEntry object, Hashtable initialization along with a set of data owned by the .NET Framework's memory management. On R_2 execution, cache-size grew rapidly as random 'in' arguments were sent with every new request, the cache-size capped at 10mb, the predefined maximum allowable cache-size, future requests replaced in-cache entries by the LRU strategy.

The CWSO State changes match expectations. The detected States match the architecture's logical design. This experiment utilized a

real-world Web Service, demonstrating general applicability of the architecture.

7. FUTURE WORK

Embed the Service Semantics Description within the service's WSDL, this maybe done by utilizing WSDL. Extensibility via attributes and element extensions. Merging the syntactic description (WSDL) with the semantic description (SSD) is very valuable, revoking the need for a separate SSD document and enabling smoother integration and richer discovery of Web Services.

8. CONCLUSION

This paper presented a generally applicable, connectivity-aware, and a transparent approach to caching of Web Services. A set of semantic tags have been identified as a prerequisite for effective web service caching. The Service Semantic Description document was developed, along with a tool enabling SSD-specification from within a widely used developer IDE. Preliminary evaluations of the architecture demonstrated general applicability, transparent operation and low resource overhead.

9. REFERENCES

- [1] D. Barbar, T. Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environments (1994).
- [2] Denis Conan, Sophie Chabridon, Olivier Villin, and Guy Bernard. Domin: A Platform for Weak Connectivity and Disconnected CORBA Objects on Hand-Held Devices (May 2003).
- [3] Douglas B. Terry and Venugopalan Ramasubramanian. Caching XML Web Services (May 2003).
- [4] Jen-Yao Chung, Kwei-Jay Lin, Richard G. Mathieu. Web Services Computing: Advancing Software Interoperability (2003).
- [5] Jia Wang. A Survey of Web Caching Schemes for the Internet (1999).
- [6] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enhancing the web infrastructure - from caching to replication (April 1997).
- [7] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, J. Schiller. Performance Considerations for Mobile Web Services (2003).
- [8] Matt Powell. XML Web Service Caching Strategies (April 2002).
- [9] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A study of integrated prefetching and caching strategies (June 1995).
- [10] Roy Friedman. Caching Web Services in Mobile Ad-Hoc Networks: Opportunities and Challenges (2002).
- [11] Sam Clegg. Reducing the Network Overheads of .NET Remoting through Runtime Call Aggregation (2003).
- [12] W3C Web Services Activity (<http://www.w3.org/2002/ws/>)
- [13] Paper Appendix (<http://www.cs.usask.ca/~kac501/880/>)

Best Available Copy

How to Turn a GSM SIM into a Web Server *Projecting Mobile Trust onto the World Wide Web*

Scott Guthery, Roger Kehr, Joachim Posegga

Scott Guthery, Mobile-Mind, sguthery@mobile-mind.com

Roger Kehr, Deutsche Telekom Research, Roger.Kehr@Telekom.de

Joachim Posegga, Deutsche Telekom Research, Joachim.Posegga@Telekom.de

Key words: Smart Cards, GSM SIMs, Web Servers, Security Infrastructure

Abstract: We describe the WebSIM, an approach that integrates GSM SIMs into the Internet. The underlying idea is to implement a Web Server inside a SIM, and to allow for transparent access to it from the Internet.
The contribution of our approach is that a SIM, which is currently a security module (smart card) fitted in a GSM mobile phone, becomes also a personal security server in the Internet. Like any other server in the Internet, it speaks TCP/IP and processes HTTP requests, e.g. for accessing certain SIM services (e.g. authentication) via CGI scripts.
The Internet connectivity of a SIM inside a mobile phone can be achieved by having a proxy host tunnel IP packets to the SIM over SMS.

If we couldn't predict the Web, what good are we?
Bob Lucky, Vice President Bellcore, 1995

1. INTRODUCTION

Much of today's wireless Internet excitement is focused on the opportunities created by pushing the World Wide Web out onto mobile telephone networks. Micro-browsers and WAP handsets seek to turn the mobile telephone into a downscale laptop computer. Relatively little thought

has gone into wondering what the mobile telephone network can bring to the Web.

Secure, reliable authentication, which is a basic prerequisite for billing customers for services on a large scale, still has no globally-accepted solution. Various attempts have been made to provide the required security technology for the Internet, but none of them has widely succeeded so far. All approaches have in practice either been considered as too insecure, or too hard to establish at the user's side. With its strong similarity to the ubiquitous credit card, a smart card is a compelling component but the required infrastructure for smart card-based solutions has been found to be hard and costly to set up.

GSM, on the other hand, provides a widely used security infrastructure, in the form of symmetric keys distributed in subscriber identity modules (SIMs). More than 250 million GSM subscribers carry around these reduced size smart cards in their mobile phones. Mobile phones can thus be seen as "wireless card readers", with the add-on of providing an I/O channel to a user for applications running inside the SIM.

The theme of the current work is that while the Web is bringing its content to the mobile phone, the mobile phone can bring its trust to the Web. The idea is to provide the authentication and authorisation capabilities from the GSM SIM to Web-based applications in a Web-friendly way; viz. as a Web server. Such a WebSIM, like any other server in the Internet, speaks TCP/IP and is transparently accessible from Internet hosts via HTTP. Specific services offered by server-enabled SIMs, for example authentication, can be implemented on the SIM using CGI scripts.

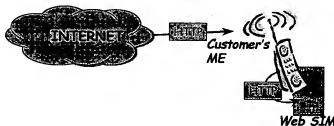


Figure 1. HTTP Requests to a SIM

Technically, this is achieved by implementing a Web server inside the GSM SIM and allowing for HTTP requests to this SIM and HTTP responses from it. Seen from the GSM perspective, this HTTP server provides selected parts of the existing application programming interface of a GSM SIM to the Internet. This makes communicating with a SIM residing in a mobile phone

identical to communicating with any Web server running in the Internet (cf. Figure 1), and the SIM can be transparently accessed from the Internet via HTTP, e.g. for authentication purposes.

2. THE WEBSIM: TECHNICAL DETAILS

A GSM SIM is an operator-trusted security server in GSM, performing computations on behalf of the GSM subscriber. The idea behind our approach, the WebSIM, is to extend the GSM SIMs into the Internet, by having them to understand a widely-used Internet protocol, HTTP, and to allow for transparent access over HTTP from the Internet. This means a GSM SIM residing in a mobile phone is not only a security server in the GSM network, but also becomes available as a security server in the World Wide Web, acting again on behalf of the GSM subscriber. We refer to such a SIM as a WebSIM.

A WebSIM, like any other server in the Internet, speaks TCP/IP and processes HTTP requests. Technically, this can be achieved by implementing a small, stripped-down Web server in a GSM SIM and making the SIMs accessible from the Internet. In this way, communicating with a SIM in a mobile phone, e.g. for authenticating a customer in the Internet, is the same as communicating with any Web server running in the Internet.

Seen from the GSM perspective, the idea behind the WebSIM is to make the interface of today's GSM SIMs (ETSI GSM 11.11 and GSM 11.14) partially available on the Internet.

2.1 The SIM's Web Server

Running a Web server in a SIM is less of a problem than one might think, in fact such servers for standard (non-GSM) smart cards were introduced in [Rees 99]. Clearly, a Web server in a SIM is not expected to host large amounts of information or HTML documents, but to provide a convenient interface to services of the SIM. These services, most of which will probably be security-related, can then be accessed via the standard protocol of the Web, HTTP, and implemented as server-side scripts on the SIM.

It is explicitly not an objective to implement versions of standard Internet services and protocols that are in full compliance with the specifications that define them. While fully compliant implementations of existing standards are certainly desirable, we are willing to give a little on full compliance, implementing only a strict subset of the specification, in order to realise an efficient yet useful smart card implementation. This design philosophy could

be summarised as "It's not how well the dog sings but that the dog sings at all."

A stripped-down version of the HTTP 1.0 protocol, which just covers the absolutely necessary part and only allow for one connection at a time, can be easily implemented with an application of less than 10K bytes inside the SIM. In particular, we can very elegantly implement this functionality as an applet on top of a GSM SIM Toolkit platform (ETSI GSM 02.19, 03.19) and then use the Toolkit's interpreter for server-side scripting. This also allows for interacting with the user of the SIM's mobile phone, since SIM Toolkit provides also an appropriate API for I/O (GSM 11.14).

2.2 Networking

Once we have an HTTP-server running in the SIM we need to connect it to the Internet. An elegant approach would be to see the mobile phone as a gateway router that passes IP packets to the SIM. If we do not want to assign a separate IP address to the SIM, we could also configure it as a process listening to port 80 on the mobile phone.

This approach, while technically elegant and easy to integrate into technologies like GPRS, requires modification of the handset and the creation of a new ETSI standard. Even if such a standard could be agreed to in a timely fashion, significant market penetration by compliant equipment would take at least two to three years. We therefore propose another approach, which can be implemented using today's protocols and mobiles.

The innovation barrier of the ME can be circumvented by a solution very common in the Internet; viz. a proxy server. We set up a proxy for the SIM on the Web and have this proxy tunnel HTTP packets through SMS to the SIM (cf. *Figure 2*). SMS messages arrive directly in the SIM and can be processed as required, e.g. by having Toolkit Applets register for such SMs. Thus, we circumvent the handset by using existing protocols and standards.

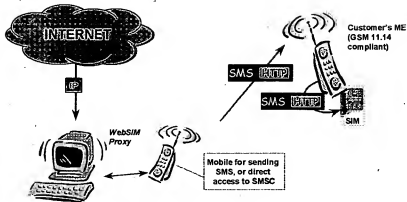


Figure 2. HTTP Tunnelling over SMS

The procedure for proxy-based IP-communication with the SIM over SMS is as follows:

1. An Internet host sends an HTTP request to the SIM's proxy.
2. The proxy embeds the request in a specially tagged SMS and sends it to the SIM.
3. The SIM passes the incoming SMS-encapsulated HTTP packet to the SIM that has registered to handle such tagged SMS.
4. The HTTP packet is extracted processed by the Web server in the SIM.
5. The HTTP response is embedded in SMS again and sent back to the proxy.
6. The proxy extracts HTTP response from the SMS and sends it back to the client in the Internet that sent the request.

As a result, the SIM can be transparently accessible by TCP/IP and HTTP from any Internet host. TCP/IP de-capsulation is handled by the proxy and the HTTP payload passed to the SIM in an SMS message. The response from the SIM is re-encapsulated by the proxy and returned to the Internet.

The proxy server approach also has some additional advantages since it can:

- implement a firewall between the Internet and the GSM network.
- guard against denial of service attacks.
- perform address translation (NAT) between the Internet address of a SIM card and the GSM address of the handset which holds the SIM.

- perform accounting and billing for WebSIM services
- eliminate the need for implementing a TCP/IP stack in the SIM

When used in practice, the WebSIM processes HTTP-bearing IP packets, for example, an URL request such as

`http://websim.dtrd.de/+49000000000/sign=(2A49C01...)`

This HTTP request can initiate the signing of the data in brackets in the SIM of the named GSM phone. After processing the request, which might consist of running other SIM-internal applications or commands, the result is sent back to the originating Internet host. Thus, integrating the capabilities of the GSM SIM into Internet applications is just like communication with any other Web server on the Internet.

2.3 Implementation

We have implemented a prototype of a WebSIM and its proxy that allows access a few SIM services over HTTP. The proxy's name is websim.dtrd.de¹, the SIM is identified by its phone number. The current implementation provides access to the following services:

- a) `http://websim.dtrd.de/+49000000000/info`
Returns information about LAI and LAC of the SIM (GSM 11.14 PROVIDE LOCAL INFO)
- b) `http://websim.dtrd.de/+49000000000/si=(item1,item2,item3,...)`
Prompts the user of the phone with a GSM 11.14 SELECT ITEM command offering the choices listed in brackets, separated by ",". Each item is interpreted as a string, the overall length of all strings must not exceed 120 bytes. The user's choice is returned.
- c) `http://websim.dtrd.de/+49000000000/gi=(prompt)`
Runs the GSM11.14 GET INPUT command and returns the text that has been entered.
- d) `http://websim.dtrd.de/+49000000000/dt=(text*)`
Runs the GSM11.14 DISPLAY TEXT with the argument supplied.

¹ Access to the server is restricted.

- e) [http://websim.dtrd.de/+49000000000/sign=\(abcdef0123456789\)](http://websim.dtrd.de/+49000000000/sign=(abcdef0123456789))
Encrypts the argument (interpreted as a string of hexadecimal characters) and returns the result.

The length of the arguments (given in brackets) is, for the sake of simplicity, restricted to fit into one SMS.

2.3.1 Proxy Implementation

The proxy is a Linux laptop running an Apache Web server with a couple of CGI scripts. These CGI scripts (implemented in Perl) take an incoming HTTP request, embed it in an SMS message, and send it off to the specified phone number.

Sending of SMS is done through a GSM mobile that is attached to the laptop with a PCMCIA modem card, which is used for sending and receiving SMS². Short messages are sent by turning the modem into TPDU mode (GSM 07.05), using a tag that causes the message to go directly to the Web Server application in the destination SIM (cf. GSM 03.48 and GSM 11.14).

Receiving SMS messages is detected by a separate process looping on the laptop that continuously polls the attached mobile for incoming short messages which are responses to pending HTTP requests. If an incoming short message is detected, it is fetched, the HTTP response is extracted and TCP/IP encapsulated and returned to the Internet client that sent the corresponding request.

2.3.2 Web Server Applet

The Web server runs as an applet on the SIM Toolkit platform (GSM 03.19) in a Schlumberger Simera SIM. The applet is written in Java, its size is currently about 7K bytes of Java byte codes. For the sake of simplicity the applet makes the following restrictions:

- a) an HTTP-Request must not exceed one SMS, and one SMS can only contain one request.
- b) the card handles only one request at a time, i.e. there is no session management inside the card.

Both restrictions can be easily overcome if needed.

We did not space-optimize the applet code at all; and we believe that it can be stripped down to a size of about 5K bytes. Noteworthy, adding new commands to the server Applet does not significantly increase its size: HTTP

² A more efficient variant would be to connect the proxy directly to the network's short message service centre (SMSC) which is the store-and-forward point for all SMS messages.

can be seen as a general-purpose application launching protocol, and once the basic HTTP-handling functionality is implemented, adding an extra command increases the applet only slightly: the difference between an applet providing only the SELECT ITEM command, and the version handling the five commands above is only a few hundred bytes.

2.3.3 Example Request

The HTTP-request `http://websim.dtrd.de/+49000000000/info` results in the following response (without HTTP headers):

LAI: 262 01
LAC: 730C

262 is the country code (Germany), 01 denotes the network (D1-Telekom), and 730C is the Local Area Code (Karlsruhe, Germany).

Overall processing time depends largely on SMS transport time, which is usually between 5 and 20 seconds one way. The proxy needs 3-4 seconds to send and receive the short messages, the handset and SIM-internal processing takes roughly another 5-6 seconds. So, the complete processing of such an HTTP request takes typically about 30 seconds.

3. APPLICATIONS OF THE WEBSIM

The WebSIM is not an application per se: it opens up the SIM to the Internet and provides an Internet-compliant interface to SIM services. Thus, it is a horizontal technology (more precisely a middleware) that supports "dot com" style applications.

We sketch a few of these applications for different domains below. Much more is possible, and in fact the most promising aspect is that the WebSIM a very convenient middleware for integrating it into Internet applications. Rather than having to deal with a different interface to a SIM each time, Internet applications can access and activate these applets in their own language, HTTP.

3.1 Provision of a secure I/O channel for Electronic commerce

Assume a customer of an Internet bookshop ordered a book for US\$ 20. When ordering it, the phone number of a WebSIM phone was provided, and the Internet bookshop can now launch a simple HTTP-request such as

[http://www.../49000.../si=\(***Bookshop,Confirm%20USD20,Cancel,...\)](http://www.../49000.../si=(***Bookshop,Confirm%20USD20,Cancel,...))



Figure 3. Screen Shot

This would result in a menu being displayed on the mobile phone of the customer (cf. Figure 3), and the Internet merchant would have established a relatively secure I/O channel to its customer (or whoever is using the customer's phone).

The security of this I/O channel can easily be enhanced by connecting over HTTPS to the proxy, or by cryptographic means as we will see next.

3.2 Authentication on the Internet

Assume Alice wants to authenticate Bob over the Internet, Oscar is Bob's GSM operator that issued a WebSIM SIM_B to Bob. Consider the following basic skeleton of a protocol:

1. $B \rightarrow A$: +490000000 [Bob's telephone number]
2. $A \rightarrow SIM_B$: [http://www.oscar.com/+490000000/sign\(RAND\)](http://www.oscar.com/+490000000/sign(RAND))
3. $SIM_B \rightarrow A$: hash(K_i ; RAND, IMSI)
4. $A \rightarrow O$: [https://www.oscar.com/verify\(RAND, hash\(\$K_i\$; RAND, IMSI\), +490000000\)](https://www.oscar.com/verify(RAND, hash(K_i; RAND, IMSI), +490000000))
5. $O \rightarrow A$: yes/no

In step 1 Bob gives Alice the phone number of his SIM. Alice then sends a random number (challenge) to the SIM_B in step 2. The SIM_B returns a hash of a secret key K_i , the random number RAND, and the SIM's IMSI to Alice in step 3.

³ IMSI = International Mobile Subscriber Identity number.

Alice can now send the response she got in step 3 to Oscar the operator, who can verify the result. Oscar knows the hash algorithm, the secret key K_i , and he can associate the phone number with the IMSI that was used in the encryption. Note that nobody besides Oscar needs to know K_i and hash. Note also that the messages of step 4 and 5 should be sent over a secure channel, e.g. by using `https://...` between Alice and Oscar.

Such a protocol can be easily refined to meet various needs one has for authentication, like including an explicit conformation from Bob, or a time stamp (for instance taken from the SMS), etc. It is a classical challenge/response authentication which can be applied to many scenarios (home banking, access control, etc.), or it can be easily adapted to provide, for instance, a session key for other purposes. For security reasons, the scenario can also be used with keys other than K_i , or with a key derived from K_i .

Essentially, this scenario is based on the principle that the mobile phone can be used as a "wireless" card reader containing a authentication token, and the security infrastructure of GSM can be easily accessed from the Internet.

3.3 Physical Access

Another nice example for using a WebSIM is the following one. Assume pushing your doorbell at home results in a request such as:

`http://websim.dtrd.de/+49000000/si=(Open,Call Intercom,Cancel)`

If you are standing in front of your door and have pushed the doorbell button, you will of course select "Open" on the menu appearing on your phone. If not, you can select to be connected with your home's internal intercom or simply cancel the request if you don't want to be disturbed.

In the case that you are connected to the intercom (which is in turn bridged to your mobile handset) you can converse with your visitor and if the situation warrants it, open the door remotely even if you aren't at home.

3.4 Handset Configuration

There are a number of SIM services local to the mobile handset that would be much easier to handle with Web interfaces. For example, the management of the SIM-internal phone book, which can be very

conveniently, updated using a Web browser if the WebSIM understands HTTP POST-methods.

4. CONCLUSION

We have presented the WebSIM, an approach integrates the GSM security module (SIM) into the Internet. The underlying idea is to integrate an HTTP server into a GSM SIM, allowing Internet connections to be made to it. This turns the SIM, which is security server for the GSM subscriber, into a WebSIM, which is a general-purpose security server for the subscriber on the Internet.

Such a WebSIM, like any other server in the Internet is transparently accessible from Internet hosts via TCP/IP and HTTP. Specific services offered by SIMs, e.g. authentication, can be accessed using CGI scripts from Internet hosts.

Seen from the GSM perspective, this HTTP server extends parts of the existing external interface of today's GSM SIMs into the Internet and is seamlessly reachable from the Internet, acting as a security server for a GSM subscriber.

The main contribution of our approach is to provide the function of SIMs in Internet-compliant protocols anyone can use. This means that the barriers for smart card applications today, 1) the lack of integration of smart cards into information technology architectures and the World Wide Web and 2) the complex interaction with cards using APDUs, is overcome by providing a simple, standard protocol, i.e. HTTP, for accessing SIM services.

A WebSIM can be accessed from anywhere on the Web using familiar Web protocols, and application programmers do not need to cope with smart card-specific interfaces any more. As with other servers in the Internet, the WebSIM processes HTTP-bearing IP packets. For example, an URL request such as

<http://websim.dtrd.de/+491710000000/sign=2A49C01>

would initiate the identity application running in the WebSIM of the named GSM phone.

After processing the request, which might consist of running other SIM-internal applications or commands, the result is sent back to the originating Internet host. Thus, integrating SIM-based security into Internet applications involves the same programming techniques as communicating with any other Web server on the Internet.

Sales of mobile handsets are forecast to soon exceed sales of personal computers. While desktop and laptop computers are great for viewing art at the Louvre, when it comes to taking action, the mobile handset with its trust-bearing SIM not to mention its portability may prove to be the primary Web surfing device. WebSIM enables the mobile to add value to the Web as opposed to simply receiving content from it.

5. REFERENCES

- [Barber 99] J. Barber, "The Smart Card URL Programming Interface," Gemplus Developer Conference, June 21 - 22, CNIT Conference Center, Paris-La Defense, France. 1999.
- [Becker 99] C.B. Becker, B. Patil and E. Qaddoura, IP Mobility Architecture Framework, IETF-Draft, October, 1999.
- [Campbell 99] A. Campbell, J. Gomez, C-Y. Wan, Z. Turanyi, and A. Valko, Cellular IP, IETF-Draft, October, 1999.
- [Comer 95] D. Comer, Internetworking with TCP/IP, Prentice Hall, 1995.
- [Di Giorgio 99] Rinaldo Di Giorgio, An introduction to the URL programming interface, Java World, September 1999.
- [GSM 02.19] Digital cellular telecommunications system (Phase 2+; Release 98): Subscriber Identity Module Application Programming Interface (SIM API); Service description; Stage 2. European Telecommunications Standards Institute, Sophia Antipolis, France. Unpublished Draft. 1999.
- [GSM 03.19] Digital cellular telecommunications system (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card™; Stage 2. European Telecommunications Standards Institute, Sophia Antipolis, France. Unpublished Draft. 1999.
- [GSM 03.60] Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Service description; Stage 2. European Telecommunications Standards Institute, Sophia Antipolis, France. Available from <http://www.etsi.org/>. 1999.
- [GSM 11.11] European digital cellular telecommunications system (Phase 2); Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface (GSM 11.11). European Telecommunications Standards Institute, Sophia Antipolis, France. Available from <http://www.etsi.org/>. 1998.
- [GSM 11.14] European Digital cellular telecommunications system (Phase 2+); Specification of the SIM application toolkit for the Subscriber Identity Module-Mobile Equipment (SIM-ME) interface (GSM 11.14). European Telecommunications Standards Institute, Sophia Antipolis, France. Available from <http://www.etsi.org/>. 1998.

- [Gustafsson 99] E. Gustafsson, A. Jonsson, E. Hubbard, J. Malmkvist, Requirements on Mobile IP from a Cellular Perspective, IETF-Draft, June, 1999.
- [Guthery 00] S. Guthery, J. Posegga, Y. Baudoin, J. Rees, "IP and ARP over ISO 7816-3", IETF Internet-Draft, February, 1, 2000.
- [Rees 99] Jim Rees and Peter Honeyman: Webcard: a Java Card web server. CITI Technical Report 99-3, Center for Information Technology Integration, University of Michigan. October 1999. [http://www.citi.umich.edu/projects/sinciti/smart card/webcard/citi-tr-99-3.html](http://www.citi.umich.edu/projects/sinciti/smart%20card/webcard/citi-tr-99-3.html)
- [Reichenbach 98] Martin Reichenbach, Herbert Damker, Hannes Federrath, Kai Rannenberg: Individual Management of Personal Reachability in Mobile Communication, pp. 163-174 in Louise Yngström, Jan Carlsen: Information Security in Research and Business; Proceedings of the IFIP TC11 13th International Information Security Conference (SEC '97): 14-16 May 1997, Copenhagen, Denmark; Chapman & Hall, London, 1998 Herbert Damker, Ulrich Pordesch, Kai Rannenberg, Michael Schneider: Aushandlung mehrseitiger Sicherheit: der Erreichbarkeits- und Sicherheitsmanager; In: Günter Möller, Kurt-Hermann Stapf: Mehrseitige Sicherheit in der Kommunikationstechnik - Erwartung, Akzeptanz, Nutzung; Addison-Wesley-Long-man, 1998.
- [RFC 1945] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, IETF RFC 1945, May, 1996.
- [RFC 2396] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, IETF RFC 2396, August, 1998.
- [RFC 2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, Leach, and T. Berners-Lee: Hypertext Transfer Protocol – HTTP/1.1, June, 1999
- [Vaha-Sipila 99] A. Vaha-Sipila, URLs for GSM Short Message Service, IETF-Draft, May 19, 1999.

This Page is inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☒ BLURED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLORED OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REPERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images
problems checked, please do not report the
problems to the IFW Image Problem Mailbox**

**A Database of Computer Attacks for the Evaluation of Intrusion
Detection Systems**

by

Kristopher Kendall

Submitted to the Department of Electrical Engineering and Computer Science in partial
fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

© Kristopher Kendall, MCMXCVIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper
and electronic copies of this thesis document in whole or in part, and to grant others the
right to do so.

Author
Department of Electrical Engineering and Computer Science,
May 21, 1999

Certified by.....
Richard Lippmann
Senior Scientist, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

*This work was sponsored by the Department of Defense Advanced Research Projects Agency
(DARPA). Opinions, interpretations, conclusions, and recommendations are those of the author
and are not necessarily endorsed by DARPA.

A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems

by
Kristopher Kendall

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 1999

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

The 1998 DARPA intrusion detection evaluation created the first standard corpus for evaluating computer intrusion detection systems. This corpus was designed to evaluate both false alarm rates and detection rates of intrusion detection systems using many types of both known and new attacks embedded in a large amount of normal background traffic. The corpus was collected from a simulation network that was used to automatically generate realistic traffic—including attempted attacks.

The focus of this thesis is the attacks that were developed for use in the 1998 DARPA intrusion detection evaluation. In all, over 300 attacks were included in the 9 weeks of data collected for the evaluation. These 300 attacks were drawn from 32 different attack types and 7 different attack scenarios. The attack types covered the different classes of computer attacks and included older, well-known attacks, newer attacks that have recently been released to publicly available forums, and some novel attacks developed specifically for this evaluation.

The development of a high quality corpus for evaluating intrusion detection systems required not only a variety of attack types, but also required realistic variance in the methods used by the attacker. The attacks included in the 1998 DARPA intrusion detection evaluation were developed to provide a reasonable amount of such variance in attacker methods. Some attacks occur in a single session with all actions occurring in the clear, while others are broken up into several sessions spread out over a long period of time with the attacker taking deliberate steps to minimize the chances of detection by a human administrator or an intrusion detection system. In some attacks, the attacker breaks into a computer system just for fun, while in others the attacker is interested in collecting confidential information or causing damage. In addition to providing detailed descriptions of each attack type, this thesis also describes the methods of stealthiness and the attack scenarios that were developed to provide a better simulation of realistic computer attacks.

Thesis Supervisor: Richard Lippmann
Title: Senior Scientist, MIT Lincoln Laboratory

Acknowledgments

First, I would like to thank my thesis advisor, Rich Lippmann, for keeping my thesis on track and for providing me with many valuable ideas and insights. I would also like to thank Rob Cunningham and Dave Fried for taking the time to read and comment on early drafts of this thesis. I appreciate the support of Rich, Rob, Dave, Isaac Graf, Seth Webster, Dan Weber, and Jonathon Korba, and the rest of the intrusion detection evaluation staff, who made working in this group a fun and highly educational experience. Finally, I would like to thank Debbie and my parents, whose love and support fuel all of my endeavors.

Contents

Chapter 1 Introduction	8
1.1 1998 DARPA Intrusion Detection System Evaluation.....	8
1.2 The Development of Attacks for the 1998 DARPA Evaluation.....	10
1.3 Outline of the Thesis.....	10
Chapter 2 Background	12
2.1 Overview of Computer Attacks.....	12
2.2 Intrusion Detection Systems.....	13
2.3 Strategies for Intrusion Detection.....	15
Chapter 3 Simulation Network	20
3.1 Modeling an Air Force Local Area Network.....	20
3.2 Simulation Hardware and Network Topology.....	21
3.3 Simulation Software.....	23
3.3.1 Virtual Machines.....	23
3.3.2 Traffic Generation.....	24
Chapter 4 Exploits	26
4.1 Sources.....	26
4.2 Age of an Exploit.....	27
4.3 A Taxonomy for Computer Attacks.....	28
4.3.1 Privilege Levels.....	29
4.3.2 Methods of Transition or Exploitation.....	30
4.3.3 Transitions Between Privilege Levels.....	31
4.3.4 Actions.....	31
4.3.5 Using the Taxonomy to Describe Attacks.....	34
4.3.6 Examples.....	35
Chapter 5 Exploits for the 1998 DARPA Evaluation	37
Chapter 6 Denial of Service Attacks	39
6.1 Apache2 R-a-Deny(Temporary/Administrative).....	40
6.2 Back R-a-Deny(Temporary).....	42

6.3	Land R-b-Deny(Administrative).....	43
6.4	Mailbomb R-a-Deny(Administrative)	44
6.5	SYN Flood (Neptune) R-a-Deny(Temporary).....	45
6.6	Ping Of Death R-b-Deny(Temporary).....	47
6.7	Process Table R-a-Deny(Temporary).....	48
6.8	Smurf R-a-Deny(Temporary)	51
6.9	Syslogd R-b-Deny(Administrative).....	54
6.10	Teardrop R-a-Deny(Temporary).....	55
6.11	Udpstorm R-a-Deny(Administrative).....	56
Chapter 7 User to Root Attacks		58
7.1	Eject U-b-S	60
7.2	Ffbconfig U-b-S.....	63
7.3	Fdformat U-b-S	64
7.4	Loadmodule U-b-S	65
7.5	Perl U-b-S	67
7.6	Ps U-b-S.....	68
7.7	Xterm U-b-S	69
Chapter 8 Remote to User Attacks		70
8.1	Dictionary R-a-U	70
8.2	Ftp-write R-c-U.....	74
8.3	Guest R-c-U.....	75
8.4	Imap R-b-S.....	76
8.5	Named R-b-S	77
8.6	Phf R-b-U.....	78
8.7	Sendmail R-b-S.....	79
8.8	Xlock R-cs-Intecept(Keystrokes)	81
8.9	Xsnoop R-c-Intecept(Keystrokes)	83
Chapter 9 Probes		85
9.1	Ipsweep R-a-Probe(Machines)	86
9.2	Mscan R-a-Probe(Known Vulnerabilities).....	87
9.3	Nmap R-a-Probe(Services).....	88
9.4	Saint R-a-Probe(Known Vulnerabilities).....	90

9.5	Satan R-a-Probe(Known Vulnerabilities).....	93
Chapter 10 Realistic Intrusion Scenarios		95
10.1	Attack Scenarios	95
10.1.1	Cracker	96
10.1.2	Spy.....	96
10.1.3	Rootkit.....	96
10.1.4	Http Tunnel	97
10.1.5	SNMP Monitoring.....	98
10.1.6	Multihop	98
10.1.7	Disgruntled/Malicious User	99
Chapter 11 Stealthiness and Actions		100
11.1	Avoiding Detection of Denial of Service (R-Deny).....	100
11.2	Avoiding Detection of Probes (R-Probe).....	101
11.3	Avoiding Detection of User to Root (L-?-S) Attacks	102
11.4	Avoiding Detection of Remote to Local (R-?-L) Attacks	106
11.5	Actions	107
Chapter 12 Attack Planning and Data Collection		109
12.1	Planning and Keeping Track of Attacks	109
12.2	Numbers of Attacks	110
12.3	Verification of Attack Success	111
Chapter 13 Results and Future Work		113
13.1	Results of the 1998 Evaluation	113
13.2	Future Work.....	116
13.2.1	Inclusion of Windows NT and Other Systems in the Evaluation.....	116
13.2.2	Better Automation of Attack Planning and Verification.....	116
13.2.3	Improved Stealthy Attacks	117
Appendix A Attack Schedule for Test Dataset		118

List of Figures

2-1	Approches to Intrusion Detection.....	16
3-1	Simulation Network Topology.....	22
4-1	Vulnerability Decreases as Time Passes.....	27
4-2	Summary of Possible Types of Actions.....	32
4-3	A Summary of Possible Attack Descriptions.....	34
5-1	The Attacks Used in the 1998 DARPA Intrusion Detection Evaluation.....	37
6-1	Summary of Denial of Service Attacks.....	40
6-2	The Internet is Used to Amplify a Ping Flood and Create a Smurf Attack.....	52
6-3	UDPStorm is Triggered by a Single Spoofed Packet.....	57
7-1	Summary of User to Root Attacks.....	59
7-2	C Code for the Eject Exploit.....	61
8-1	Summary of Remote to Local Attacks.....	71
8-2	Plot of Connections to Pop3 Service During a Dictionary Attack.....	72
8-3	Illustration of Sendmail Attack.....	80
9-1	Summary of Probes.....	85
9-2	Plot of Connections During a Medium Level Saint Scan.....	91
9-3	Plot of Connections During a Medium Level Satan Scan.....	94
10-1	Attacker Uses Http to Tunnel Through a Firewall.....	98
11-1	Attacker Can Use "at" to Temporally Disassociate the Time of Access and Time of Attack.....	104
11-2	Transcript of a Stealthy Eject Attack.....	105
12-1	Instances of Non-Scenario Attacks.....	111
13-1	Current Intrusion Detection Systems Cannot Find New DoS and Remote to User Attacks.....	114
13-2	Clear vs Stealthy Attack Detection Rates.....	115

Chapter 1

Introduction

1.1 1998 DARPA Intrusion Detection System Evaluation

Heavy reliance on networked computer resources and the increasing connectivity of these networks has greatly increased the potential damage that can be caused by attacks launched against computers from remote sources. These attacks are difficult to prevent with firewalls, security policies, or other mechanisms because system and application software is changing at a rapid pace, and this rapid pace often leads to software that contains unknown weaknesses or bugs. Intrusion detection systems are designed to detect those attacks that inevitably occur despite security precautions. Some intrusion detection systems detect attacks in real time and can be used to stop an attack in progress. Others provide after-the-fact information about attacks that can be used to repair damage, understand the attack mechanism, and reduce the possibility of future attacks of the same type [43].

Many parties are working on the development of intrusion detection systems, including universities, commercial software companies, and organizations within the Department of Defense. As these groups explore different methods and develop various new systems for intrusion detection, it is clearly advantageous to have a means of evaluating the success of these systems in detecting attacks. The best environment for

testing and evaluation of an intrusion detection system is the actual environment in which it will be used. However, research groups often do not have access to operational networks on which to test their systems, and these systems (especially while they are still in early development) are tested in a simulated environment. The ability to perform accurate testing and evaluation in a simulated environment requires high-quality data that is similar to the traffic (including attacks) that one finds on operational networks. In general, this data is difficult to acquire because it contains private information and reveals potential vulnerabilities of the networks from which the data is collected. These factors led to DARPA sponsorship of MIT Lincoln Laboratory's 1998 intrusion detection evaluation, which created the first standard corpus for the evaluation of intrusion detection systems.

The 1998 intrusion detection evaluation was the first of an ongoing series of yearly evaluations conducted by MIT Lincoln Laboratory under DARPA ITO and Air Force Research Laboratory sponsorship. These evaluations contribute significantly to the intrusion detection research field by providing direction for research efforts and calibration of current technical capabilities. The 1998 evaluation was designed to be simple, to focus on core technology issues, and to encourage the widest possible participation by eliminating security and privacy concerns and by providing data types that are used by the majority of intrusion detection systems. Data for the first evaluation was made available in the summer of 1998. The evaluation itself occurred towards the end of the summer. A follow-up meeting for evaluation participants and other interested parties was held in December 1998 to discuss the results of the evaluation.

1.2 The Development of Attacks for the 1998 DARPA Evaluation

This thesis describes the computer attacks that were included in the 1998 DARPA intrusion detection evaluation. A large sample of actual computer attacks was needed to accurately test the performance of intrusion detection systems. These attacks needed to cover the different classes of attack types. Many of the attacks used in the evaluation were drawn from public sources, but some novel attacks were developed specifically for use in this evaluation. In all cases, these attacks had to be adapted to work reliably in the largely automated simulation network from which the 1998 DARPA evaluation data were collected. Later sections of this thesis discuss the methods that were developed to create realistic simulations of computer intrusion scenarios, and the methods that were developed to vary the degree of attack stealthiness. People who attack computer networks often have goals beyond simply gaining access to a system. Some attackers break into computers simply for the challenge, others are interested in collecting information, and some are motivated by the desire to cause damage. Attackers also vary in their level of sophistication, and an accurate evaluation of intrusion detection systems requires testing how well the systems are able to detect attacks from all types of attackers—from the relative novice who is not aware that an intrusion detection system is monitoring a network, to the sophisticated, experienced cracker who knows about intrusion detection systems and takes steps to avoid being caught.

1.3 Outline of the Thesis

Chapter 2 presents background information about computer attacks and intrusion detection systems. This provides perspective for later discussion on the development of

attacks for use in evaluating these systems. The different types of computer attacks and some of the many intrusion detection strategies that are currently being developed are discussed.

Chapter 3 describes the simulation network that was used to collect data for the 1998 DARPA intrusion detection evaluation. This network consisted of 11 computers and one router that, with the aid of software developed for use in the evaluation, simulated a large network consisting of hundreds of machines, and thousands of users.

Chapters 4, 5, 6, 7, 8, and 9 focus on the different types of attacks that were developed for use in the evaluation. Chapter 4 presents a taxonomy of computer attacks that was useful in choosing attack types to include in the evaluation. Chapter 5 is an introduction to Chapters 6 through 9 which discuss in detail the specific attacks within each of the four broad categories of attacks included in the evaluation (denial of service attacks, attacks that give a local user superuser access, attacks that give a remote user local access, and probes).

Chapters 10 and 11 discuss the attack scenarios and stealthy methods that were used to create a more realistic simulation of actual computer attacks.

Chapter 12 describes the processes that were used to plan, collect, and verify the attack instances that were included in the 1998 DARPA intrusion detection evaluation.

Finally, in Chapter 13 the results of the 1998 DARPA intrusion detection evaluation are summarized and discussed with a focus on the implications of these results for both the intrusion detection research community, and the Lincoln Laboratory group that will be conducting future evaluations.

Chapter 2

Background

2.1 Overview of Computer Attacks

In its broadest definition, a computer attack is any malicious activity directed at a computer system or the services it provides. Examples of computer attacks are viruses, use of a system by an unauthorized individual, denial-of-service by exploitation of a bug or abuse of a feature, probing of a system to gather information, or a physical attack against computer hardware. A subset of the possible types of computer attacks were included in the 1998 DARPA intrusion detection system evaluation, including: (1) Attacks that allow an intruder to operate on a system with more privileges than are allowed by the system security policy, (2) Attacks that deny someone else access to some service that a system provides, or (3) Attempts to probe a system to find potential weaknesses.

The following paragraphs provide some examples of the many ways that an attacker can either gain access to a system or deny legitimate access by others.

Social Engineering: An attacker can gain access to a system by fooling an authorized user into providing information that can be used to break into a system. For example, an attacker can call an individual on the telephone impersonating a network administrator in an attempt to convince the individual to reveal confidential

information (passwords, file names, details about security policies). Or an attacker can deliver a piece of software to a user of a system which is actually a trojan horse containing malicious code that gives the attacker system access.

Implementation Bug: Bugs in trusted programs can be exploited by an attacker to gain unauthorized access to a computer system. Specific examples of implementation bugs are buffer overflows, race conditions, and mishandled of temporary files.

Abuse of Feature: There are legitimate actions that one can perform that when taken to the extreme can lead to system failure. Examples include opening hundreds of telnet connections to a machine to fill its process table, or filling up a mail spool with junk e-mail.

System Misconfiguration: An attacker can gain access because of an error in the configuration of a system. For example, the default configuration of some systems includes a "guest" account that is not protected with a password.

Masquerading: In some cases it is possible to fool a system into giving access by misrepresenting oneself. An example is sending a TCP packet that has a forged source address that makes the packet appear to come from a trusted host.

2.2 Intrusion Detection Systems

Intrusion detection systems gather information from a computer or network of computers and attempt to detect intruders or system abuse. Generally, an intrusion detection system will notify a human analyst of a possible intrusion and take no further action, but some newer systems take active steps to stop an intruder at the time of detection [49].

Although there are many possible sources of data an intrusion detection system can use, three types of data were provided to participants in the 1998 Lincoln Laboratory intrusion detection evaluation. Most intrusion detection systems in existence today use one or more of these three types of data. The first of these data sources is traffic sent over the network. All data that is transmitted over an ethernet network is visible to any machine that is present on the local network segment. Because this data is visible to every machine on the network, one machine connected to this ethernet can be used to monitor traffic for all the hosts on the network. During the DARPA evaluation, network traffic was sniffed using a single machine running the tcpdump program [39] to save the network traffic. A second source of data for an intrusion detection system is system-level audit data. Most operating systems offer some level of auditing of operating system events. The amount of data that is collected could be as limited as logging failed attempts to log in, or as verbose as logging every system call. Basic Security Module (BSM) [62] data from a Solaris victim machine was collected and distributed as part of the DARPA evaluation data. A third source of data distributed to the evaluation participants was information about file system state. Daily file system dumps were collected from each of the machines used in the simulation. An intrusion detection system that examines this file system data can alert an administrator whenever a system binary file (such as the ps, login, or ls program) is modified. Normal users have no legitimate reason to alter these files, so a change to a system binary file indicates that the system has been compromised. Although there are many other potential sources of data that can be used by an intrusion detection system to find attacks (such as real-time process lists, logfiles, processor loads, etc.), these three sources (sniffed network traffic,

host-level audit files, and file-system state) were provided to participants in the 1998 Lincoln Laboratory DARPA intrusion detection evaluation because they were determined to be the sources most commonly used by the evaluation participants.

After the three types of data were collected and aggregated, the data was distributed to participants via CD-ROM. Once participants obtained this data, each group used its particular intrusion detection system to find intrusions and abuses that were inserted into the collected traffic. Although the 1998 DARPA evaluation tested only the ability to find attacks offline, some intrusion detection systems can evaluate data in real-time, allowing administrators (or the system itself) to take defensive action against the intruder.

2.3 Strategies for Intrusion Detection

The different approaches that have been pursued to develop intrusion detection systems are described in many papers, including [3][44][63]. Figure 2-1 shows four major approaches to intrusion detection and the different characteristics of these approaches. The lower part of this figure shows approaches that detect only known attacks, while the upper part shows approaches that detect novel attacks. Simpler approaches are shown on the left and approaches that are both computationally more complex and have greater memory requirements are shown towards the right.

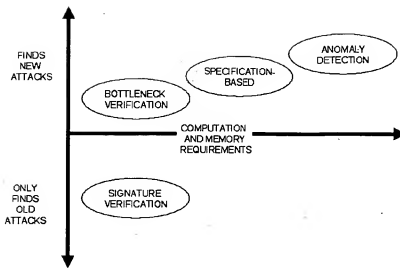


Figure 2-1: Approaches to Intrusion Detection

The most common approach to intrusion detection, denoted as “signature verification” is shown on the bottom of Figure 2-1. Signature verification schemes look for an invariant sequence of events that match a known type of attack. For example, a signature verification system that is looking for a Ping Of Death denial-of-service attack (an oversize ping packet that causes some machines to reboot) would have a simple rule that says “any ping packet of length greater than 64 kilobytes is an attack.” Attack signatures can be devised that detect attempts to exploit many possible system vulnerabilities, but a large drawback of this strategy is that it is difficult to establish rules that identify novel types of attacks. The Network Security Monitor (NSM) was an early signature-based intrusion detection system that found attacks by searching for keywords in network traffic captured using a sniffer. Early versions of the NSM [29][28] were the foundation of many government and commercial intrusion detection systems, including

NetRanger [23] and NID [41]. Signature verification systems are popular because one sniffer can monitor traffic to many workstations and the computation required to reconstruct network sessions and search for keywords is not excessive. In practice, these systems can have high false-alarm rates (e.g. 100's of false-alarms per day) because it is often difficult to select keywords by hand that successfully detect real attacks without creating false alarms for normal traffic. In addition, signature verification schemes must be updated frequently to detect new attacks as they are discovered. Recent research on systems which rely on signature verification includes BRO [47] and NSTAT [38].

The approaches shown in the upper half of Figure 2-1 can be used to find novel attacks. This capability is essential to protect critical hosts because new attacks and attack variants are constantly being developed.

Anomaly detection, shown in the upper right of Figure 2-1, is one of the most frequently suggested approaches to detect novel new attacks. Anomaly detection schemes construct statistical models of the typical behavior of a system and issue warnings when they observe actions that deviate significantly from those models. NIDES was one of the first statistical-based anomaly detection systems used to detect unusual user [36] and unusual program [1] behavior. The statistical component of NIDES forms a model of a user, system, or network activity during an initial training phase. After training, anomalies are detected and flagged as attacks. Of course, anomalous behavior does not always signal that an attack is taking place, so anomaly detection systems need to be carefully tuned to avoid high false alarm rates. This level of tuning is only possible if normal user or system activity is stable over time and does not overlap with attacker activity. A user with very regular habits will be easy to model, and

any intruder attempting to masquerade as such a user would likely exhibit behavior that deviated significantly from the user's normal activity. The actions of a system administrator, however, might be more irregular and harder to distinguish from the actions of an attacker. In addition, a hacker may be able to slowly change the characteristics that an anomaly detection system considers "normal" by deviating only slightly from normal behavior over a long period of time. After the anomaly detection system had been trained to consider more actions "normal" the attacker could mount an attack and avoid detection. A second disadvantage of anomaly detection schemes is the large computation and memory resources required to maintain the statistical model. Recent research on anomaly detection includes the development of EMERALD [46], which combines statistical anomaly detection from NIDES with signature verification.

Specification-based intrusion detection [39] is a second approach that can be used to detect new attacks. It detects attacks that make improper use of system or application programs. This approach involves first writing security specifications that describe the normal intended behavior of programs. Host-based audit records are then monitored to detect behavior that violates the security specifications. This approach was applied to 15 UNIX system programs and successfully found many attacks [39]. Specification-based intrusion detection has the potential to provide very low false alarm rates and detect a wide range of attacks including many forms of malicious code such as trojan horses, viruses, attacks that take advantage of race conditions, and attacks that take advantage of improperly synchronized distributed programs. Unfortunately, it is difficult to apply because security specifications must be written for all monitored programs. This is difficult because system and application programs are constantly updated. Specification-

based intrusion detection is thus best applied to a small number of critical user or system programs that might be considered prime targets for an attack.

The final strategy shown in Figure 2-1 is bottleneck verification. The bottleneck verification approach applies to situations where there are only a few, well defined ways to transition between two groups of states. One example of such a well-defined transition is transitions from a normal user to superuser within a shell. If an individual is in the user state, the only way to legally gain root privileges is by using the su command and entering the root password. Thus, if a bottleneck verification system can detect a shell being launched, determine the permissions of the new shell, and detect the successful use of the su command to gain root access (or, more importantly, the lack of a successful su command), then illegal transitions from normal user to root user can be detected—even if the transition is being made through some novel method that did not exist when the bottleneck verification system was created [65].

Chapter 3

Simulation Network

The goal of the 1998 DARPA Intrusion Detection System Evaluation was to collect and distribute the first standard corpus for evaluation of intrusion detection systems. This corpus was designed to evaluate both false alarm rates and detection rates of intrusion detection systems using many types of both known and novel attacks embedded in a large amount of normal background traffic. One roadblock that has discouraged the creation of such a corpus is the reluctance of companies and government agencies to release data collected from operational computer networks. Data collected from an operational computer network is optimal for the evaluation of intrusion detection systems, but this data may contain personal or sensitive information that could not be released to the many parties who conduct intrusion detection research. For this reason, all data in the 1998 DARPA Intrusion Detection System evaluation was synthesized and recorded on a network which *simulated* an operational network connected to the Internet [25][42].

3.1 Modeling an Air Force Local Area Network

The goal of the simulation network (or simnet) was to accurately simulate the network traffic of a Local Area Network that one might find at a United States Air Force facility. Automatically generated traffic used more than 20 network services, including dns, finger, ftp, http, ident, ping, pop, smtp, snmp, telnet, time, and X. In order to accurately

model the features of an Air Force network, statistics were measured from months of actual network traffic that was collected from more than fifty Air Force computer networks. Traffic statistics for automatically generated traffic matched average Air Force statistics. The content of email, ftp, web sites, and files was similar to actual documents. Content was generated using open-source documentation or statistical reconstruction from a large set of unclassified documents that preserved both unigram word frequencies and also the frequency of two-word sequences. Generating the data from public sources allowed the data to be distributed without security or privacy concerns.

The 32 different types of attacks that were inserted in the data collected on the simulation network represent attacks that one would expect to see on an Air Force network comprised mostly of UNIX workstations. These attacks were chosen to represent a mixture of different categories of computer attacks, and to provide a mix of older, more recent, and even some novel attacks. In some cases, transcripts of actual Air Force intrusions were used to develop attack scenarios. When such transcripts were not available, attacks were chosen from publicly known attacks and accounts of intrusions on civilian computer systems. Later sections of this thesis describe the attacks in detail.

3.2 Simulation Hardware and Network Topology

The simulation network consisted of two Ethernet network segments connected to each other through a router. A diagram of the network topology is shown in Figure 3-1. The router is located in the top center of this picture and is labeled "CISCO". Everything to the left of the router in the figure is considered the "outside" of the network and everything to the right of the router is the "inside" of the simulation network. The inside network is connected to one interface of the router and consists of all the computers that

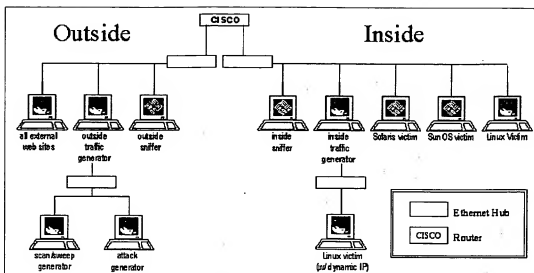


Figure 3-1: Simulation Network Topology

are part of the fabricated “eyrie.af.mil” domain. The computers that model the rest of the Internet are connected to the external interface of the router.

The simulation network included eleven computers. The outside of the network contains a traffic generator (for both background traffic and automated attacks), a web server, a sniffer/recorder, and two machines used for non-automated attack generation. The inside of the network (eyrie.af.mil) consisted of a background traffic generator, a sniffer/recorder, a Solaris 2.5 victim, a SunOS 4.1.4 victim, a Linux 4.2 victim and a Linux 5.0 victim that could dynamically change IP addresses. Although two computers (one inside, one outside) generated all of the background traffic in the simulation, a modification to the operating system of these computers enabled them to act as hundreds of “virtual” machines [26]. The same modification was made to the outside web server so this machine could mimic thousands of web servers. The modification to the operating system that created these virtual machines will be discussed in the next section.

3.3 Simulation Software

The hardware for the simulation network consists of only eleven machines and one router. Custom software allows this simple network to model the interaction of thousands of clients and servers. Without such software the simulation network would consist of hundreds of computers and the evaluation would be impractical in terms of both the cost and time required to maintain the network hardware.

3.3.1 Virtual Machines

The two traffic generators on the simulation network were configured with a modification to the Linux kernel provided by the Air Force Research Laboratory [26]. The IP swapping kernel allows each process that is started on the traffic generator to specify which IP address should be used as the source address for all network traffic generated by that process. At the time of invocation of a new process, an entry is added to a table (called the IP swap table) that maps process id to IP address. Whenever a network packet is generated, the kernel checks the process id of the process that generated the packet and uses the entry in the IP swap table as the source address of the new packet. The modified kernel was an important step in creating the illusion of a virtual machine, but there are several other modifications that support the creation of a single host that can act as many hosts. Daemons that provide network services—including telnetd, ftpd, and login—display banners that identify the machine they are running on. These daemons were modified to give different identification information depending on the destination IP address specified in the request.

3.3.2 Traffic Generation

Automatic traffic regeneration was used to create automated network sessions containing both artificial normal traffic and attacks that appear to be generated by humans working on a system. The traffic regeneration mechanism was designed to be:

- Automatic, requiring no human intervention.
- Reproducible; when repeated, sessions produce identical results.
- Robust, so it can run for long periods without human supervision.

As suggested in [48], the *expect* language was used to automate interaction between a system and a user, allowing autonomous sessions to be run as if a user were typing at a keyboard. Unfortunately, *expect* cannot be used to automate the interaction between a user and a graphical environment (such as X Windows), so any sessions that required interaction with a graphical user interface were run by human actors during the simulation. Because the amount of interaction included in the simulation was so large, creating a separate *expect* script for each session would have been unwieldy. For this reason, an *expect* program called the “regenerator” (for “traffic regenerator”) was written that reads information about a particular session from a specially formatted “exs” file. Each “exs” file contains a header and a body. Within the header is information about what time to start a session, the IP address of the local machine, the IP address of the remote machine, and a list of prompts we expect to see during this session. The body of the “exs” file contains one or more prompt-response pairs. For each prompt-response pair the regenerator first waits for the prompt indicated, pauses an indicated amount of time, and then types the command. Each session within the simulation is represented by a single “exs” script. To simulate a day of traffic, a day of “exs” scripts can be collected

and passed to the regenerator which then runs all of the sessions. If an error occurs in generating or collecting this traffic, the traffic collection can be repeated by rerunning the regenerator with the same "exs" scripts. The regenerator satisfies the need for automatic, reproducible, and robust traffic generation.

Chapter 4

Exploits

A large sample of actual computer attacks is needed to test an intrusion detection system. These attacks should cover the different classes of attack types and contain exploits for both newly discovered as well as older well-known vulnerabilities. An attack instance might consist of several phases. For example, an attacker might copy a program onto a system, run this program that exploits a system vulnerability to gain root privileges, and then use this root privilege to install a backdoor into the system for later access. This chapter provides some background on the different types of computer exploits, including a taxonomy that can be used to organize these exploits into meaningful groups.

4.1 Sources

Many of the exploits developed for the 1998 DARPA evaluation were drawn from ideas or implementations available from public sources on the Internet. Rootshell [50] is a web-site dedicated to collecting computer exploits and has a sizeable archive of attacks for many popular operating systems. The “Bugtraq” mailing list also frequently hosts “exploit code”—ostensibly released for the purpose of testing one’s own system for vulnerability—when a new vulnerability is discussed. A searchable archive of the Bugtraq mailing list can be found on the Internet at <http://www.geek-girl.com>. Other exploits were created from information released by computer security groups such as

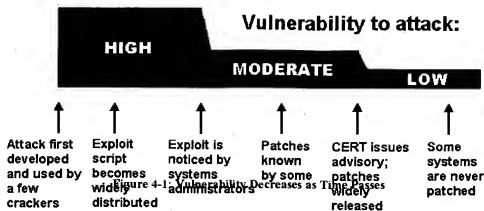


Figure 4-1: Vulnerability to a Newly Discovered Attack Decreases with Time

CERT [22] and ISS X-force [35] that frequently release information about new vulnerabilities. Additional sources of information about system vulnerabilities and possible exploits were vendor-initiated bulletins posted by operating system vendors like Sun Microsystems and Redhat Software. These bulletins are released to customers to encourage them to download patches that eliminate a new vulnerability.

New and novel exploits were also created specifically for the purpose of the evaluation. These new exploits are useful for determining how well an intrusion detection system works against novel attacks that were not publicly known at the time the intrusion detection system was developed.

4.2 Age of an Exploit

Each new exploit has a period of time during which it is most dangerous. Figure 4-1 is a simplified illustration of the various phases of a newly discovered exploit. The vertical axis of the figure shows vulnerability to attack and the horizontal axis represents time. As time progresses, more people are made aware of the vulnerability and patch their

systems to make them resistant to the exploit. Even after news of a particular vulnerability has become widespread, some systems might not be patched. Some computer systems with less-experienced administrators go years without having widely-known security holes fixed. Some of these older attacks were included in the set of attacks used for the 1998 DARPA evaluation. Intrusion detection systems were generally able to find these older, well-known attacks.

4.3 A Taxonomy for Computer Attacks

A taxonomy for classifying computer attacks was used to choose exploits for the evaluation. A good taxonomy makes it possible to classify attacks into groups that share common properties. Once these groups have been identified, the job of adequately testing an intrusion detection system becomes easier because instead of developing every possible attack we can choose a representative subset from each group. The taxonomy presented here was originally presented in [64]. The features of this taxonomy are:

- Each attack can be reliably placed in one category.
- All possible intrusions have a place in the taxonomy
- The taxonomy can be extended in the future.

This taxonomy was created for the express purpose of testing and evaluating intrusion detection systems. Within the taxonomy, each attack can be categorized as one of the following:

- A user performs some action at one level of privilege
- A user makes an unauthorized transition from a lower privilege level to a higher privilege level.

- A user stays at the same privilege level, but performs some action at a higher level of privilege.

The taxonomy requires a way of describing each privilege level, a way to describe transitions, and a way of categorizing actions. These three requirements are presented in the following sections.

4.3.1 Privilege Levels

The taxonomy defines an approach to ranking levels of privilege. The privilege categories that are applied within this thesis are:

R	Remote network access
L	Local network access
U	User access
S	Root/Super-user access
P	Physical Access to Host

Having privilege at the “Remote network access” level refers to having, via an interconnected network of systems, minimal network access to a target system. “Local network access” represents the ability to read from and write to the local network that the target machine uses. “User access” refers to the ability to run normal user commands on a system. “Root/Superuser access” gives a user total software control of a system. “Physical Access to Host” allows the operator to physically manipulate characteristics of the system (i.e. remove disk drives, insert floppy disks, turn the system off). This list only represents a subset of all possible access levels, but these were the most useful categories for describing the attacks in the 1998 DARPA intrusion detection evaluation.

4.3.2 Methods of Transition or Exploitation

An attacker needs to exploit some failure of a security framework in order to perform an attack. The five methods of transition that were explored for the 1998 DARPA evaluation and the single letters (**m,a,b,c,s**) used to represent the methods were:

- m) Masquerading:** In some cases it is possible to fool a system into giving access by misrepresenting oneself. Examples of masquerading include using a stolen username/password or sending a TCP packet with a forged source address.
- a) Abuse of Feature:** There are legitimate actions that one can perform, or is even expected to perform, that when taken to the extreme can lead to system failure. Example include filling up a disk partition with user files or starting hundreds of telnet connections to a host to fill its process table.
- b) Implementation Bug:** A bug in a trusted program might allow an attack to proceed. Specific examples include buffer overflows and race conditions.
- c) System Misconfiguration:** An attacker can exploit errors in security policy configuration that allows the attacker to operate at a higher level of privilege than intended.
- s) Social Engineering:** An attacker may be able to coerce a human operator of a computer system into giving the attacker access.

An individual attack may use more than one of these methods. For example, a bug in the implementation of the TCP stack on some systems makes it possible to crash the system by sending it a carefully constructed malformed TCP packet. This packet may also have

the source address forged so as to avoid identification of the attacker. Such an attack would be exploiting both masquerading and an implementation bug, and it would be possible to detect the intrusion by noting either of these features.

4.3.3 Transitions Between Privilege Levels

To show a transition between two privilege levels the strings for the two levels are written adjacent to one another with the method of transition between them. Two examples are shown in the following table:

Examples:

<i>Attack</i>	<i>String</i>	<i>Description</i>
Format	U-b-S	User exploits a bug in the format program to become root/superuser
Ftp-write	R-c-U	A user with remote network access exploits a badly configured anonymous ftp server to gain local user access

4.3.4 Actions

There are many actions that can occur as part of a computer attack. Within the taxonomy, actions are represented with a string that represents a category, and a specification string that describes the specific action taken. For example, the string **Probe(Users)**, represents some action taken by an attacker to gather information about the users of a system. The following paragraphs describe the five categories of actions that were used to describe the actions taken during the 1998 DARPA intrusion detection evaluation.

<i>Category</i>	<i>Specific Type</i>	<i>Description</i>
Probe	Probe(Machines)	Determine types and numbers of machines on a network
	Probe(Services)	Determine the services a particular system supports
	Probe(Users)	Determine the names or other information about users with accounts on a given system
Deny	Deny(Temporary)	Temporary Denial of Service with automatic recovery
	Deny(Administrative)	Denial of Service requiring administrative intervention
	Deny(Permanent)	Permanent alteration of a system such that a particular service is no longer available
Intercept	Intercept(Files)	Intercept files on a system
	Intercept(Network)	Intercept traffic on a network
	Intercept(Keystrokes)	Intercept keystrokes pressed by a user
Alter	Alter(Data)	Alteration of stored data
	Alter(Intrusion-Traces)	Removal of hint of an intrusion, such as entries in log files
Use	Use(Recreational)	Use of the system for enjoyment, such as playing games or bragging on IRC
	Use(Intrusion-Related)	Use of the system as a staging area/entry point for future attacks

Figure 4-2: Summary of Possible Types of Actions.

Probes are actions taken by an attacker to gather information about one or more machine. Probes are represented within the taxonomy by the category label of “**Probe**”. The specific types of probes used in the DARPA evaluation were: (1) Probing a network to see how many and what types of machines are on that network (**Probe(Machines)**), (2) Probing a system to see what services the system supports (**Probe(Services)**), and (3) Probing a system to find out information about user accounts on that system (**Probe(Users)**).

Denial of service attacks are attempts to interrupt or degrade a service that a system provides. These attacks are represented within the taxonomy by the category label **"Deny"**. The classes of denial of service attacks used in the DARPA evaluation were: (1) Temporary denial of service with automatic recovery (**Deny(Temporary)**), (2) Denial of service requiring administrative action for recovery (**Deny(Administrative)**), and (3) Permanent denial of service with total system reconstruction required for recovery (**Deny(Permanent)**).

Another category of attacker actions is the interception of data. Interception of data is represented within the taxonomy by the category label **"Intercept"**. The types of data interception actions used in the 1998 DARPA evaluation were: (1) Interception/Reading of files on a file system (**Intercept(Files)**), and (2) Interception of packets on a network (**Intercept(Network)**).

An additional category of action is the alteration or creation of data on a system or network. Actions that involve data alteration or creation are represented with the category label **"Alter"**. The types of data alteration used in the 1998 DARPA evaluation were: (1) Alteration of data stored on a system, such as a password file or any other file (**Alter(Data)**), and (2) Removal of hints of an intrusion, such as entries in log files (**Alter(Intrusion-Traces)**).

The final category of attacker action described in the taxonomy is "use" of a system. Any use of the system that does not fall into the categories described above can be placed in the category represented by the category label **"Use"**. The specific ways in which an attacker might use a system that were included in the 1998 DARPA evaluation were: (1) Use of the system by the intruder for enjoyment or recreational purposes such

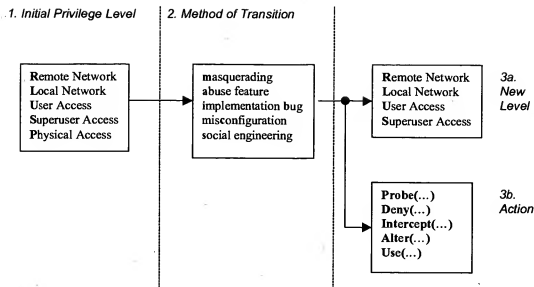


Figure 4-3: A Summary of Possible Attack Descriptions

as playing games or bragging on IRC (**Use(Recreational)**), and (2) Use of a system as a staging ground or entry point for attacks on other systems (**Use(Intrusion-Related)**).

The action categories and specifications described in the previous paragraphs are summarized in Table 4-2. Each row of this table represents a specific type of action within a category. These specific actions have been grouped according to the categories presented above. The first column of the table is the action category, the second column is the string that represents the action in the taxonomy, and the third column in each row is a description of that particular type of action.

4.3.5 Using the Taxonomy to Describe Attacks

Figure 4-3 gives an overview of the approach the taxonomy uses for the classification of attacks. Each attack classified by this taxonomy is represented as a short alpha-numeric string. The sub-strings in this description are the bold strings from the previous three

sections. The initial privilege level is indicated by **R**, **L**, **U**, **S**, or **P** as defined in section 4.3.1, the actions are indicated by any of the strings presented in section 4.3.4, and the method of exploitation is indicated by **m**, **a**, **b**, **c**, or **s** as defined in section 4.3.2.

In order to describe an attack, select the privilege level that the attacker had before the attack occurred, from the possible choices of **Remote Network**, **Local Network**, **Local User**, **Superuser/Root**, and **Physical Access**. Next, select the method of exploitation, if the method is known. If the method is unknown, then a question mark ("??") is used to indicate the method of exploitation. The possible choices are **masquerading**, **abuse of feature**, **implementation bug**, **misconfiguration**, or **social engineering**. Finally, either indicate the level of privilege the attacker gained as a result of the exploit (again with **R**, **L**, **U**, **S**, or **P**) or the actions the user performed at the current level of privilege. The taxonomy presented in [64] has additional features that are not discussed here, but the material presented in the section provides a subset of the taxonomy that is useful for the purpose of discussing the attacks in the 1998 DARPA intrusion detection evaluation.

4.3.6 Examples

The following table presents three examples that show the correct formatting of the alphanumeric string that specifies an action being performed at a specific privilege level. In a SYN flood (or neptune) attack the attacker sends a stream of SYN packets to a port on a target machine. For a short period of time after these packets have been sent, other users are unable to access the network services provided by that port. In the second example a user runs the crack program to decrypt the password file of a machine that has been compromised. In the third example, the attacker uses the Ffbconfg attack to make a

transition from Local User privilege to Root/Superuser privilege, and then uses this new privilege level to alter the password file on the victim system.

Examples:

<i>Attack</i>	<i>String</i>	<i>Description</i>
SYN flood	R-a-Deny(temporary)	A user with remote network access temporarily denies service
Cracking passwords	U-Use(Intrusion)	A user with a local account runs a program which attempts to decrypt entries in the password file.
ffbconfig	U-b-S-Alter(Files)	An attacker with a local account uses a bug in the Ffbconfig program to gain root access and alter files.

Chapter 5

Exploits for the 1998 DARPA Evaluation

Figure 5-1 shows the 32 different exploits that were used in the 1998 DARPA intrusion detection evaluation. This table presents the attacks broken up into categories of type and

Figure 5-1: The Attacks Used in the 1998 DARPA Intrusion Detection Evaluation

	Solaris	SunOS	Linux
Denial Of Service (R-Deny)	Apache2 Back Mailbomb Neptune Ping Of Death Process Table Smurf Syslogd UDP Storm	Apache2 Back Land Mailbomb Neptune Ping of death Process Table Smurf UDP Storm	Apache2 back Mailbomb Neptune Ping of death Process Table Smurf Teardrop UDP Storm
Remote to User (R-?-U)	dictionary ftp-write guest phf xlock xsnoop	dictionary ftp-write guest phf xlock xsnoop	dictionary ftp-write guest imap named phf sendmail xlock xsnoop
User to Super-user (U-?-S)	eject fribconf fidformat ps	loadmodule ps	perl xterm
Surveillance/ Probing (R-Probe)	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan

vulnerable operating system. The four type categories represent groupings of the possible attack types listed in the taxonomy. These four groups are: Denial of Service (R-?-Deny), Remote to Local User (R-?-U), Local User to Super-user (U-?-S), and Probes (R-?-Probe). The three columns of the table divide the exploits by target platform. Some attacks are listed in more than one column. The Smurf attack, for example, is listed three times—in the Solaris column, the SunOS column, and the Linux column—because all three operating systems are vulnerable to the Smurf attack. The next four chapters present detailed descriptions of each class of attack, and the individual attacks from that class that were included in the 1998 DARPA intrusion detection evaluation

Chapter 6

Denial of Service Attacks

A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of denial of service (or DoS) attacks. Some DoS attacks (like a mailbomb, neptune, or smurf attack) abuse a perfectly legitimate feature. Others (teardrop, Ping of Death) create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others (apache2, back, syslogd) take advantage of bugs in a particular network daemon. Figure 6-1 provides an overview of the denial of service attacks used in the 1998 DARPA intrusion detection evaluation. Each row represents a single type of attack. The six columns show the attack name, a list of the services that the attack exploits, the platforms that are vulnerable to the attack, the type of mechanism that is exploited by the attack (implementation bug, abuse of feature, masquerading, or misconfiguration), a generalization of the amount of time the attack took to implement, and a summary of the effect of the attack. The following sections describe in detail each of the Denial of Service attacks that were included in the 1998 DARPA intrusion detection evaluation.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Apache2	http	Any Apache	Abuse	Short	Crash httpd
Back	http	Any Apache	Abuse/Bug	Short	Slow server response
Land	N/A	SunOS	Bug	Short	Freeze machine
Mailbomb	smtp	All	Abuse	Short	Annoyance
SYN Flood	Any TCP	All	Abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None	Bug	Short	None
Process Table	Any TCP	All	Abuse	Moderate	Deny new processes
Smurf	icmp	All	Abuse	Moderate/Long	Network Slowdown
Syslogd	syslog	Solaris	Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Bug	Short	Reboot machine
Udpstorm	echo/ chargen	All	Abuse	Short	Network Slowdown

Figure 6-1: Summary of Denial of Service Attacks

6.1 Apache2

R-a-Deny(Temporary/Administrative)

Description

The Apache2 attack is a denial of service attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash [4].

Simulation Details

This exploit was adapted from C code originally posted to the bugtraq mailing list. A C-shell wrapper was also created which executes the apache2 C program in a loop until the server being attacked is no longer responsive.

As soon as the attack was launched the load average (as reported by the “top” program) of the victim server jumped to 5 or more. As more and more requests were submitted to the web server the memory usage and load average of the victim continued to climb until eventually the httpd daemon ran out of memory and crashed. At this point the server no longer responded to http requests and the httpd daemon needed to be restarted by the superuser for service to be restored.

Attack Signature

Every http request submitted as part of this exploit contains many http headers. Although the exact number and value of these headers could be varied by an attacker, the particular version of the exploit which was used in the 1998 DARPA evaluation sent http GET requests with the header “User-Agent: sioux\r\n” repeated 10000 times in each request. The actual content of the header is not important for the exploit—the exploit is only dependent on the fact that http request contains *many* headers. A typical http request contains twenty or fewer headers, so the 10000 headers used by this exploit are quite anomalous.

6.2 Back

R-a-Deny(Temporary)

Description

In this denial of service attack against the Apache web server, an attacker submits requests with URL's containing many frontslashes. As the server tries to process these requests it will slow down and becomes unable to process other requests [55].

Simulation Details

The Back attack was implemented as a C shell script that used the Netcat [31] tool to generate network traffic. This shell script was adapted from a script originally posted to the Bugtraq mailing list. Although the number of frontslashes in the URL sent by the shell script could be varied, the number of frontslashes that was determined to be optimal for denial of service against Apache running on Linux 4.2 was between six and seven thousand.

The Back attack causes instances of the httpd process on the victim to consume excessive CPU time. This consumption of the CPU slows down all the system's activities, including responses to network requests. The system recovers automatically when the attack stops.

Attack Signature

An intrusion detection system looking for the Back attack needs to know that requests for documents with more than some number of frontslashes in the URL should be considered an attack. Certainly, a request with 100 frontslashes in the URL would be highly irregular on most systems. This threshold could be varied to find the desired balance between detection rate and false alarm rate.

6.3 Land

R-b-Deny(Administrative)

Description

The Land attack is a denial of service attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS 4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [17].

Simulation Details

The land exploit program used in the DARPA evaluation was adapted from a C implementation found at <http://www.rootshell.com>. The exploit is quite simple and the code could easily be rewritten in any language with access to the TCP sockets interface. The code sends a single SYN packet with the source address spoofed to be the same as the destination address.

Within the simulation, this exploit was run against a Sun SPARC WorkStation running SunOS version 4.1. When a TCP SYN packet with an identical source and destination address was received by this host, the system completely locked up. In order to restore service, the machine had to be physically turned off and on again.

Attack Signature

The Land attack is recognizable because IP packets with identical source and destination addresses should never exist on a properly working network.

6.4 Mailbomb

R-a-Deny(Administrative)

Description

A Mailbomb is an attack in which the attacker sends many messages to a server, overflowing that server's mail queue and possibly causing system failure.

Simulation Details

This exploit was implemented as a perl program that constructed mail messages and connected to the SMTP port of the victim machine directly. The Mailbomb perl program accepted as parameters the e-mail addresses of victims as well as the number of e-mail messages to send.

Although the Mailbomb exploit was used several times throughout the simulation with different parameters, a typical attack would send 10,000 one kilobyte messages (10 megabytes of total data) to a single user. This volume of messages was not enough to adversely effect the performance of the server or cause system failure. As implemented, this attack was more of a nuisance for a particular user than a real threat to the overall security of a server.

Attack Signature

An intrusion detection system that is looking for a mailbomb attack can look for thousands of mail messages coming from or sent to a particular user within a short period of time. This identification is a somewhat subjective process. Each site might have a different definition of how many e-mail messages can be sent by one user or to one user before the messages are considered to be part of a mailbomb.

6.5 SYN Flood (Neptune)

R-a-Deny(Temporary)

Description

A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the “tcpd” server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative [13].

Simulation Details

The neptune exploit code used in the simulation was compiled from C code originally posted to the bugtraq archive. The neptune program allows the user to specify a victim host, the source address to use in the spoofed packets, the number of packets to send, and the ports to hit on the victim machine (including an “infinity” option that would attack all ports).

The neptune exploit was effective against all three of the victim machines used in the simulation. Every TCP/IP implementation is vulnerable to this attack to a varying degree depending on the size of the data structure used to store incoming connections and the

timeout value associated with half-open connections. As a point of reference, sending twenty SYN packets to a port on a Solaris 2.6 system will cause that port to drop incoming requests for approximately ten minutes. During the simulation, a neptune attack which sent 20 SYN packets to every port from 1 to 1024 of the Solaris server once every ten minutes was able block incoming connections to any of these ports for more than an hour.

Attack Signature

A neptune attack can be distinguished from normal network traffic by looking for a number of simultaneous SYN packets destined for a particular machine that are coming from an unreachable host. A host-based intrusion detection system can monitor the size of the tcpd connection data structure and alert a user if this data structure nears its size limit.

6.6 Ping Of Death

R-b-Deny(Temporary)

Description

The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting [14].

Simulation Details

Several implementations of the Ping of Death exploit can be found at <http://www.rootshell.com> as well as many other sources on the web. This exploit is popular because early versions of the ping program distributed with Microsoft Windows95 would allow the user to create oversized ping packets simply by specifying a parameter at the command line (i.e. ping -l 65510). Thus, many users could potentially exploit this bug without even making the effort to download and compile a program.

The Ping of Death attack affected none of the victim systems used in the evaluation. The attack was included as an example of an attempted known attack that fails to have an effect.

Attack Signature

An attempted Ping of Death can be identified by noting the size of all ICMP packets and flagging those that are longer than 64000 bytes.

6.7 Process Table

R-a-Deny(Temporary)

Description

The Process Table attack is a novel denial-of-service attack that was specifically created for this evaluation. The Process Table attack can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the superuser can create, other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine's process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine.

An example of a service that is vulnerable to this attack is the finger service. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program's source code that must be bypassed in order to initiate a successful process attack. In a typical implementation (specifics will vary depending on the actual UNIX version used), if inetd receives more than 40 connections to a particular service within 1 minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence. To launch a successful process table attack against a computer running inetd and finger, the following sequence may be followed: 1. Open a connection to the target's finger port. 2. Wait for 4 seconds.

3. Repeat steps 1-2. This attack has been attempted against a variety of network services on a variety of operating systems. It is believed that the imap and sendmail servers are vulnerable. Most imap server software contains no checks for rapid-fire connections. Thus, it is possible to shut down a computer by opening multiple connections to the imap server in rapid succession. With sendmail the situation is reversed. Normally, sendmail will not accept connections after the system load has jumped above a predefined level. Thus, to initiate a successful sendmail attack it is necessary to open the connections very slowly, so that the process table keeps growing in size while the system load remains more or less constant [6].

Simulation Details

The version of this exploit used in the simulation was implemented as a perl script that would open connections to a port every n seconds, where the port and the number of seconds n are specified as run-time parameters. The connections were maintained until a user terminated the script.

The number of connections that must be opened before denial of service is accomplished depends on the size of the process table in the operating system of the victim machine. By using the Process Table attack on the fingerd port as described above, the process table of a Solaris server was exhausted after opening approximately 200 connections (at a rate of one connection every four seconds it took about 14 minutes before the process table was full). After the process table was full, no new process could be launched on the victim machine until the attack was terminated by the attacking program or an administrator manually killed the connections initiated by the attacking script (which is quite difficult to do without launching a new process).

Attack Signature

Because this attack consists of abuse of a perfectly legal action, an intrusion detection system that is trying to detect a process table attack will need to use somewhat subjective criteria for identifying the attack. The only clue that such an attack is occurring is an “unusually” large number of connections active on a particular port. Unfortunately “unusual” is different for every host, but for most machines, hundreds of connections to the finger port would certainly constitute unusual behavior.

6.8 Smurf

R-a-Deny(Temporary)

Description

In the "smurf" attack, attackers use ICMP echo request packets directed to IP broadcast addresses from remote locations to create a denial-of-service attack. There are three parties in these attacks: the attacker, the intermediary, and the victim (note that the intermediary can also be a victim) [18]. The attacker sends ICMP "echo request" packets to the broadcast address (xxx.xxx.xxx.255) of many subnets with the source address spoofed to be that of the intended victim. Any machines that are listening on these subnets will respond by sending ICMP "echo reply" packets to the victim. The smurf attack is effective because the attacker is able to use broadcast addresses to amplify what would otherwise be a rather innocuous ping flood. In the best case (from an attacker's point of view), the attacker can flood a victim with a volume of packets 255 times as great in magnitude as the attacker would be able to achieve without such amplification. This amplification effect is illustrated by Figure 6-2. The attacking machine (located on the left of the figure) sends a single spoofed packet to the broadcast address of some network, and every machine that is located on that network responds by sending a packet to the victim machine. Because there can be as many as 255 machines on an ethernet segment, the attacker can use this amplification to generate a flood of ping packets 255 times as great in size (in the best case) as would otherwise be possible. This figure is a simplification of the smurf attack. In an actual attack, the attacker sends a *stream* of icmp "ECHO" requests to the broadcast address of *many* subnets, resulting in a large, continuous stream of "ECHO" replies that flood the victim.

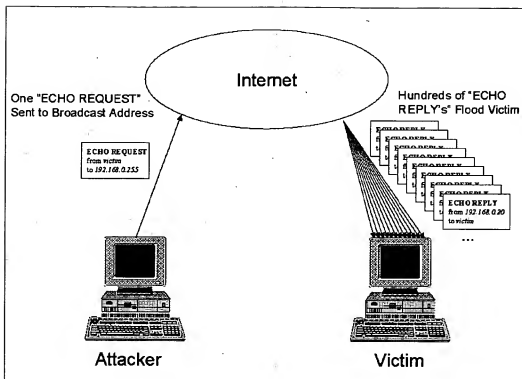


Figure 6-2: The Internet is Used to Amplify a Ping Flood and Create a Smurf Attack

Simulation Details

Because the simulation network for the 1998 DARPA evaluation has a flat network topology with only two physical subnets, the smurf attack as described above could not be implemented on the simulation network. For this reason, the “smurfsim” program was developed to recreate the observable effects of a smurf attack. Smurfsim uses the raw socket API to construct ICMP packets with forged source addresses. Smurfsim takes as parameters the IP address of the victim, the number of packets to send, the average percentage of hosts on a subnet that are alive, and a comma-separated list of subnets. The program then randomly constructs a list of hosts that are alive on each of the subnets in the comma-separated list and starts sending “echo reply” packets to the victim, that have been spoofed to look like they originating from the hosts in the list. This behavior is

exactly what would occur if an attacker had performed an actual Smurf attack in which “echo request” packets (with the source address spoofed to be that of the victim machine) were sent to the broadcast address of each subnet given in the parameter list.

Several different simulated Smurf attacks were included in the evaluation data. In the most extreme case, the smurfsim program was used to simulate a smurf attack that generated traffic from 100 subnets for a period of one hour. During this period of time the entire simulation network was unresponsive and other network sessions (such as normal users trying to send e-mail, etc) would time out before they could be completed. In all, this particular attack instance generated over two gigabytes of network packets.

Attack Signature

The Smurf attack can be identified by an intrusion detection system that notices that there are a large number of “echo replies” being sent to a particular victim machine from many different places, but no “echo requests” originating from the victim machine.

6.9 Syslogd

R-b-Deny(Administrative)

Description

The Syslogd exploit is a denial of service attack that allows an attacker to remotely kill the syslogd service on a Solaris server. When Solaris syslogd receives an external message it attempts to do a DNS lookup on the source IP address. If this IP address doesn't match a valid DNS record, then syslogd will crash with a Segmentation Fault [54].

Simulation Details

The Syslogd exploit used in the 1998 DARPA evaluation was a C program that was originally posted to the Bugtraq mailing list. This code was compiled to create an exploit program that was used to remotely cause the syslogd program to crash on a Solaris 2.5 server. Once syslogd has crashed it must be manually restarted by an administrator for the logging service to be restored.

Attack Signature

The one way to reliably recognize this attack with a network-monitoring intrusion detection system is to notice a packet destined for the syslog port that contains an unreachable source address. Of course, it may not be realistic for an intrusion detection system to check every packet destined for the syslog port to see whether or not the source address is resolvable. If no remote system logging is expected to occur on a particular network, any external syslog messages appearing on this network is likely to be an attack. Finally, a host-based intrusion detection system could be configured to notice the syslog process die because of a segmentation fault.

6.10 Teardrop

R-a-Deny(Temporary)

Description

The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments [17].

Simulation Details

The teardrop name is derived from a widely available C program that exploits this vulnerability. This exploit code can be found at <http://www.rootshell.com> and in the Bugtraq archives. Although many systems are rumored to be vulnerable to the teardrop attack, of the systems used in the DARPA evaluation, only the Redhat Linux 4.2 systems were vulnerable. The teardrop attack would cause these machines to reboot.

Attack Signature

An intrusion detection system can find this attack by looking for two specially fragmented IP datagrams. The first datagram is a 0 offset fragment with a payload of size N, with the MF bit on (the data content of the packet is irrelevant). The second datagram is the last fragment (MF = 0), with a positive offset greater than N and with a payload of size less than N [5].

6.11 Udpstorm

R-a-Deny(Administrative)

Description

A Udpstorm attack is a denial of service attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed. For example, by connecting a host's chargen service to the echo service on the same or another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. An illustration of such an attack is presented in Figure 6-2. The figure demonstrates how an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. First, the attacker forges a single packet that has been spoofed to look like it is coming from the echo port on the first victim machine and sends it to the second victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request, so when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in like kind, and the loop of traffic continues until it is stopped by intervention from an external source [10].

Simulation Details

Code that exploits this vulnerability was posted to the bugtraq mailing list. This program sends a single spoofed UDP packet to a host. This single spoofed packet is able to create a never-ending stream of data being sent from the echo port of one machine to the echo

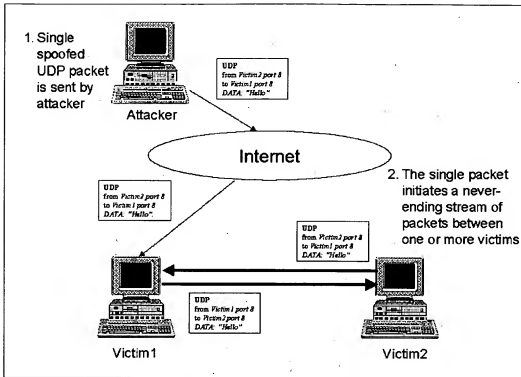


Figure 6-3: UDPStorm is Triggered by a Single Spoofed Packet

port of another. This loop created network congestion and slowdown that would continue until the inetd daemon was restarted on one of two victim machines.

Attack Signature

This attack can be identified in two ways. First, the single packet that initiates the attack can be recognized because it is a packet originating from outside the network that has been spoofed to appear as if it is coming from a machine inside the network. Second, once the loop of network traffic has been initiated, an intrusion detection system that can see network traffic on the inside of the network can note that traffic is being sent from the charge or echo port of one machine to the charge or echo port of another.

Chapter 7

User to Root Attacks

User to Root exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.

There are several different types of User to Root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. For example, if a program expects the user to input the user's first name, the programmer must decide how many characters that first name buffer will require. Assume the program allocates 20 characters for the first name buffer. Now, suppose the user's first name has 35 characters. The last 15 characters will overflow the name buffer. When this overflow occurs, the last 15 characters are placed on the stack, overwriting the next set of instructions that was to be executed. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system. Despite the fact that programmers can eliminate this problem through careful programming techniques, some common utilities are susceptible to buffer overflow attacks [2]. Another class of User to Root attack exploits programs that make assumptions about the environment in which they are running. A good example of such an attack is the

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Eject	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Ffbconfig	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Fdformat	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Loadmodule	Any user session	SunOS	Poor Environment Sanitation	Short	Root Shell
Perl	Any user session	Linux	Poor Environment Sanitation	Short	Root Shell
Ps	Any user session	Solaris	Poor Temp File Management	Short	Root Shell
Xterm	Any user session	Linux	Buffer Overflow	Short	Root Shell

Figure 7-1: Summary of User to Root Attacks

loadmodule attack, which is discussed below. Other User to Root attacks take advantage of programs that are not careful about the way they manage temporary files. Finally, some User to Root vulnerabilities exist because of an exploitable race condition in the actions of a single program, or two or more programs running simultaneously [27]. Although careful programming could eliminate all of these vulnerabilities, bugs like these are present in every major version of UNIX and Microsoft Windows available today.

Figure 7-1 summarizes the User to Root attacks used in the 1998 DARPA evaluation. Note from this table that all of the User to Root attacks can be run from any interactive user session (such as by sitting at the console, or interacting through telnet or rlogin), and that all of the attacks spawn a new shell with root privileges. The following sections describe each of the User to Root attacks that was used in the 1998 DARPA intrusion detection evaluation in greater detail.

7.1 Eject

U-b-S

Description

The Eject attack exploits a buffer overflow in the “eject” binary distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, libvolmgt.so.1, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems[60].

Simulation Details

A truncated version of the eject exploit that was used in the 1998 evaluation is shown in Figure 7-2. This exploit was originally posted to the bugtraq mailing list. The exploit script, once compiled, can be run in a command line session on a Solaris server to spawn a shell that ran with root privileges.

Attack Signature

There are several ways that an intrusion detection system might identify this attack. Assuming an attacker already has access to an account on the victim machine and is running the exploit as part of a remote session, a network-based system can look at the contents of the telnet or rlogin session the attacker is using and notice one of several features.

First, assuming that an attacker transmits the C code to the victim machine unencrypted, the intrusion detection system could look for specific features of the source code. For example, an intrusion detection system could look for the string “Jumping to

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5
6 #define BUF_LENGTH 264
7 #define EXTRA 36
8 #define STACK_OFFSET 8
9 #define SPARC_NOP 0xc013a61c
10 u_char sparc_shellcode[] =
11
12 ...
13
14 17 ";
15 18
16 19 u_long get_sp(void)
17 20 {
18 21     __asm__ ("mov %sp,%i0 \n");
19 22 }
20 23
21 24 void main(int argc, char *argv[])
22 25 {
23 26     char buf[BUF_LENGTH + EXTRA + 8];
24 27     long targ_addr;
25 28     u_long *long_p;
26 29     u_char *char_p;
27 30     int i, code_length = strlen(sparc_shellcode), dso=0;
28 31
29 32     if(argc > 1) dso=atoi(argv[1]);
30 33
31 34     long_p = (u_long *) buf;
32 35     targ_addr = get_sp() - STACK_OFFSET - dso;
33 36
34 37     for (i = 0; i < (BUF_LENGTH - code_length) / sizeof(u_long); i++)
35 38         *long_p++ = SPARC_NOP;
36
37 ...
38
39 43     for (i = 0; i < EXTRA / sizeof(u_long); i++)
40 44         *long_p++ = targ_addr;
41 45     printf("Jumping to address 0x%x B[%d] E[%d] SO[%d]\n",
42 46         targ_addr, BUF_LENGTH, EXTRA, STACK_OFFSET);
43 47     execl("/bin/eject", "eject", & buf, (char *) 0);
44 48     perror("execl failed");
45 49 }

```

Unique Strings that can be Used to Identify an Eject Attack

Figure 7-2: C code for the Eject Exploit

address” on line 45 of the source code or the line “execl (“/bin/eject” ; ‘eject’, & buf(char *) 0);” from line 47.

Even if the attacker encrypts the source code, the attack leaves a distinct signature. A segment of the transcript from an actual instantiation of this attack that was used in the simulation is shown below:

```
pascal> /tmp/162562
Jumping to address 0xffffffff B[364] E[400] SO[704]
#
```

An intrusion detection system that saw only these three lines has several clues that an attack has taken place. First, the user's prompt has changed from "pascal>" to "#" without running the su command. Second, the string "Jumping to address" is again printed. Of course, a careful attacker would remove this line from the source code, but simply looking for the string "Jumping to address" would catch the less careful attacker. Finally, a host-based intrusion detection system could catch an eject attack either by noticing the invocation of the eject program with a large argument, or by performing bottleneck verification [43] on the transition from normal user to root user and noticing that the user did not make a legal user to root transition.

7.2 Ffbconfig

U-b-S

Description

The Ffbconfig attack exploits a buffer overflow in the "fbconfig" program distributed with Solaris 2.5. The fbconfig program configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWfbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the fbconfig program [61].

Simulation Details

This attack is very similar to the eject attack described above. Once again, C code to exploit this vulnerability was posted on the Bugtraq mailing list.

Attack Signature

The means by which an intrusion detection system can identify the fbconfig attack are similar to those described above for the Eject exploit. An attacker who is exploiting this vulnerability must first transfer the code for the exploit (either C code to be compiled, or pre-compiled code) onto the victim machine, and then run the exploit. As with the eject exploit, there are strings within the source code of the fbconfig exploit script which identify the attack to a network or host based intrusion detection system. A host-based intrusion detection system can perform bottleneck verification or look for the invocation of the command `"/usr/sbin/fbconfig"` with an oversized argument for the `"-dev"` parameter.

7.3 Fdformat

U-b-S

Description

The Fdformat attack exploits a buffer overflow in the "fdformat" program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program [60].

Simulation Details

Exploit code for this vulnerability was posted to the Rootshell Website [53] in March, 1997. The exploit code was used unmodified for the DARPA evaluation.

Attack Signature

Methods for identifying this attack are nearly identical to those described for the eject and ffbconfig attacks.

7.4 Loadmodule

U-b-S

Description

The Loadmodule attack is a User to Root attack against SunOS 4.1 systems that use the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine [8].

Simulation Details

The code for the loadmodule exploit script is widely available on the internet. This code is usually in the form of a shell script—it does not need to be compiled before it is run.

The steps of the attack are quite simple:

1. Change the value of the internal field separator (or IFS) variable to a slash.
2. Add “.” To the front of the PATH variable
3. Copy “/bin/sh” to “./bin”
4. Execute “/usr/openwin/bin/loadmodule a”.

When the loadmodule shell script (which is setuid root by default) executes, it attempts to run the command “exec(‘/bin/a’);”. Since the IFS variable has been changed to “/” the string “/bin/a” is parsed into two tokens, and the loadmodule script attempts to run the first—“bin”. Since the attacker has conveniently put a copy of “/bin/sh” in the current directory and named it “bin”, the loadmodule script (which is running as root) will exec “./bin”—giving the attacker a shell with root privileges.

Attack Signature

This attack can be identified either by performing bottleneck verification with a host-based intrusion detection system, or by keyword spotting with a network based intrusion

detection system. A simple rule could say that any session which contained the strings "set \$IFS="V"" and "loadmodule" in close proximity was probably a loadmodule attack. Of course, an attacker could quite easily hide from such a simple rule. Detailed discussion of such methods for hiding is presented in Chapter 11.

7.5 Perl

U-b-S

Description

The Perl attack is a User to Root attack that exploits a bug in some Perl implementations. Suidperl is a version of Perl that supports saved set-user-ID and set-group-ID scripts. In early versions of suidperl the interpreter does not properly relinquish its root privileges when changing its effective user and group IDs. On a system that has the suidperl, or sperl, program installed and supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access [12].

Simulation Details

A perl script that uses this vulnerability to gain root access was made publicly available in August 1996 [51]. The code is only two lines long, and can easily be executed from the command-line. Once this perl script has run, the user will be presented with a new shell that is running with root privileges.

Attack Signature

The methods by which an intrusion detection system could identify a perl exploit attempt are identical to those described above for the loadmodule attack. A host-based intrusion detection system could notice that a root shell was spawned without a legal user to root transition, or a network-based intrusion detection system could look the strings “\$>=0;” or “exec (‘/bin/sh’);”, which have little valid use except in an exploit attempt.

Description

The Ps attack takes advantage of a race condition in the version of "ps" distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege. This race condition can only be exploited to gain root access if the user has access to the temporary files. Access to temporary files may be obtained if the permissions on the /tmp and /var/tmp directories are set incorrectly. Any users logged in to the system can gain unauthorized root privileges by exploiting this race condition [9].

Simulation Details

This exploit is possible because of a combination of the ps program not carefully managing temporary files and a buffer overflow. A shell script that builds a carefully constructed temporary file, creates a C-file, compiles the code and executes the exploit was found at Rootshell.com [52]. Once an attacker has transfers this shell script onto a Solaris victim machine and runs it, a root shell will be spawned for the attacker.

Attack Signature

Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

7.7 Xterm

U-b-S

Description

The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 (as well as other operating systems not used in the simulation) and allows an attacker to execute arbitrary instructions with root privilege. Problems exist in both the xterm program and the Xaw library that allow user supplied data to cause buffer overflows in both the xterm program and any program that uses the Xaw library. These buffer overflows are associated with the processing of data related to the inputMethod and preeditType resources (for both xterm and Xaw) and the *Keymap resources (for xterm). Exploiting these buffer overflows with xterm when it is installed setuid-root or with any setuid-root program that uses the Xaw library can allow an unprivileged user to gain root access to the system [21].

Simulation Details

C source code that exploits this vulnerability on Redhat Linux 5.0 systems was found at the Rootshell website [56]. Once again, an attacker can compile this C code, and when the resulting program is run the attacker is given a shell running with root privileges.

Attack Signature

Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

Chapter 8

Remote to User Attacks

A Remote to User attack occurs when an attacker who has the ability to send packets to a machine over a network—but who does not have an account on that machine—exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks discussed within this section exploit buffer overflows in network server software (imap, named, sendmail). The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies. The Xlock attack involves social engineering—in order for the attack to be successful the attacker must successfully spoof a human operator into supplying their password to a screensaver that is actually a trojan horse. Figure 8-1 summarizes the characteristics of the Remote to User attacks that were included in the 1998 DARPA intrusion detection evaluation. The following sections provide details of each of these attacks.

8.1 Dictionary

R-a-U

Description

The Dictionary attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making repeated guesses at possible usernames and passwords.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Dictionary	telnet, rlogin, pop, imap, ftp	All	Abuse of Feature	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	telnet, rlogin	All	Misconfiguration	Short	User-level access
Imap	imap	Linux	Bug	Short	Root Shell
Named	dns	Linux	Bug	Short	Root Shell
Phf	http	All	Bug	Short	Execute commands as user http
Sendmail	smtp	Linux	Bug	Long	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor Keystrokes remotely

Figure 8-1: Summary of Remote to Local Attacks

Users typically do not choose good passwords, so an attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords. Dictionary guessing can be done with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using usernames and passwords. Figure 8-2 is a plot of the connections made to the pop3 port of a victim machine during a dictionary attack that is using the pop service to check for valid login/password combinations. The horizontal axis of this plot represents time in minutes, and each line segment in the plot is a single connection to the pop3 service. Lines representing successive sessions are

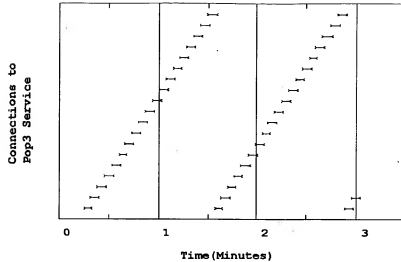


Figure 8-2: Plot of Connections to Pop3 Service During a Dictionary Attack

displaced vertically slightly and wrap around (in this figure at roughly 1.5 minutes). The length of the lines represents the length of the pop sessions. Lines begin with a greater-than sign ">" and end with a less-than sign "<", and thus form an "x" for short sessions. In all, this example dictionary attack consists of 40 attempts to log in, with a 4 second delay between each attempt.

Simulation Details

A perl script that performed automated password guessing on a variety of services was developed specifically for use in our evaluation. The "Netguess" perl script could take in a file of possible username/password combinations, or create guesses for the password based on simple permutations of the username. Within the simulation, this script was used to perform between 10 and 100 login attempts on the telnet, ftp, and pop services.

When the script was successful in gaining access to the system, it would immediately quit and report success.

Attack Signature

An intrusion detection system that finds attempted dictionary attacks needs to know the session protocol of every service that provides username/password authentication. For a given service, the intrusion detection system must be able to recognize and record failed login attempts. Once this functionality is available, detecting dictionary attacks is a matter of setting a detection threshold based on the number of failed login attempts within a given period of time.

8.2 Ftp-write

R-c-U

Description

The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system [7].

Simulation Details

This attack was implemented as an expect script which was created explicitly for use in the simulation. This expect script anonymously logged in to the ftp service on the victim machine, created a ".rhosts" file with the string "+ +" in it within the ftp home directory, disconnected from the ftp server, used rlogin to connect back to the server as user "ftp", and finally performed some actions on the victim machine. Creating a ".rhosts" file in the ftp home directory with the entry "+ +" in it allows any user from any machine to rlogin to the victim as user "ftp".

Attack Signature

An intrusion detection system can monitor for this attack by watching all anonymous ftp sessions and assuring that no files are created in the ftp root directory.

8.3 Guest

R-c-U

Description

The Guest attack is a variant of the Dictionary attack described in Section 8.1. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit [27].

Simulation Details

The Guest attack is a simplified version of the Dictionary attack discussed earlier in this chapter. The same “Netguess” perl script that was used to simulate a Dictionary attack was used to simulate the Guest attack—the only difference between the implementation of the two attacks was the command-line options that were passed to the Netguess program. Whereas the Dictionary attack would try up to a hundred user names and thousands of username/password combinations, the Guest attack would make only a couple of login attempts, using combinations such as “guest/<none>”, “guest/guest”, “anonymous/<none>” and “anonymous/anonymous”.

Attack Signature

Because the Guest attack is essentially a subset of the Dictionary attack, the methods for finding the two attacks are basically the same. An intrusion detection system that is looking for a Dictionary attack already should need only minor tuning in order to find attempts to log in to the guest account.

8.4 Imap

R-b-S

Description

The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges [16].

Simulation Details

The Imap attack used in the 1998 DARPA intrusion detection evaluation was part of the Impack 1.03 attack toolkit [34]. This toolkit contained precompiled binary programs for the Linux platform that would scan for vulnerable machines, as well as send the necessary message to exploit the buffer overflow and gain access to a root shell. The Impack contained detailed instructions on how to use these precompiled programs and took very little skill to use. The release of the Impack made this vulnerability especially dangerous, as any user with a Linux machine and the ability to follow instructions could use this attack to remotely gain root access to any vulnerable hosts.

Attack Signature

The Imap attack can be identified by an intrusion detection system that has been programmed to monitor network traffic for oversized Imap authentication strings.

8.5 Named

R-b-S

Description

The Named attack exploits a buffer overflow in BIND version 4.9 releases prior to BIND 4.9.7 and BIND 8 releases prior to 8.1.2. An improperly or maliciously formatted inverse query on a TCP stream destined for the named service can crash the named server or allow an attacker to gain root privileges [19].

Simulation Details

The version of the Named exploit used in the simulation was adapted from a C program originally posted to the Bugtraq mailing list. This program, once compiled, would connect to the named port on a victim machine and overflow a buffer of the named server with instructions that would send an xterm running with root privilege back to the attacker's X console. Because this attack involved interaction with X, all of the Named attacks included in the simulation were run by human actors.

Attack Signature

The Named attack can be identified by watching DNS inverse query requests for messages that are longer than the 4096 byte buffer allocated for these requests within the "named" server.

8.6 Phf

R-b-U

Description

The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function `escape_shell_cmd()` to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server [11].

Simulation Details

The Phf attack is quite simple to implement because it requires only the ability to connect to a network socket and issue an http request. Within the simulation, the Netcat [31] program was used to generate this http request. Although this vulnerability allows an attacker to run any command on the server, the command used throughout the simulation was `"/bin/cat /etc/passwd"`. Using the Phf attack with this command reveals the contents of the victim system's password file to users with no account on the victim machine.

Attack Signature

To find the Phf attack, an intrusion detection system can monitor http requests watching for invocations of the phf command with arguments that specify commands to be run. Examples of commands that an attacker might attempt to execute by exploiting the phf exploit are: `cat /etc/passwd`, `id`, `whoami`, or `xterm`.

8.7 Sendmail

R-b-S

Description

The Sendmail attack exploits a buffer overflow in version 8.8.3 of sendmail and allows a remote attacker to execute commands with superuser privileges. By sending a carefully crafted email message to a system running a vulnerable version of sendmail, intruders can force sendmail to execute arbitrary commands with root privilege [15].

Simulation Details

Although this vulnerability was widely known about at the time of the evaluation, no code that exploited this vulnerability had been posted to public forums such as Bugtraq, or Rootshell.com. A significant period of time (one person working for more than two weeks) was spent developing the first known implementation of this exploit explicitly for use in the evaluation. The implementation consists of a carefully constructed mail message which, when sent to the victim machine with a vulnerable version of sendmail, adds a new entry with root privilege to the end of the password file on the victim system. Once this new entry has been added, the attacker can log into the machine as this new user and execute commands as a root user. Figure 8-3 provides an illustration of an instantiation of the Sendmail attack as it was implemented in the simulation. In step 1 of this illustration, the attacker sends a carefully crafted e-mail message to the victim machine. In step 2, the sendmail daemon starts to process this message, overflows one of its buffers, and executes the attacker's inserted commands that create a new entry in the password file. In step 3, the attacker comes back to the victim machine and uses the new password file entry to gain root access to the victim machine and perform some malicious actions.

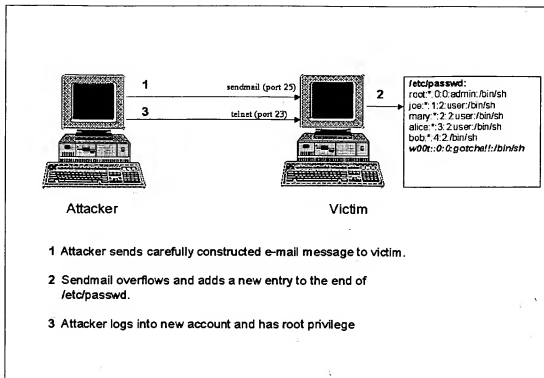


Figure 8-3: Illustration of the Sendmail Attack.

Attack Signature

The Sendmail attack overflows a buffer in the MIME decoding routine of the sendmail program. In order for an intrusion detection system to identify a Sendmail attack it must monitor all incoming mail traffic and check for messages that contain a MIME header line that is inappropriately large.

8.8 Xlock

R-cs-Intecept(Keystrokes)

Description

In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected, into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing "xhost +"), hoping to convince the user sitting at that console to type in their password. If the user sitting at the machine being attacked actually types their password into the trojan version of xlock the password will be sent back to the attacker.

Simulation Details

This attack was created specifically for use in the evaluation. A special version of the xlock program was created by making small modifications to the publicly available source code for the xlock program. The standard version of xlock will not display on a remote display, and does not save or output the password that the user enters. A modified version of xlock was created that allows the attacker to display the screen saver on a remote display and returns the password entered by the victim. Because of the required interaction with the X server, the Xlock attack was always run by a human actor.

Attack Signature

Two factors make this attack quite difficult for an intrusion detection system to identify. First, this attack is a spoofing attack that does not abuse a bug in the system, but relies on assumptions by the person who is currently using the system. Second, this attack is embedded in the X protocol, and an intrusion detection system that is trying to identify this attack must understand and parse these X communications in order to identify the

Xlock attack. One non-optimal way of dealing with these difficulties, is to program an intrusion detection system to identify as suspicious any X traffic that is originating from an unknown machine that is destined for a machine being monitored.

8.9 Xsnoop

R-c-Intercept(Keystrokes)

Description

In the Xsnoop attack, an attacker watches the keystrokes processed by an unprotected X server to try to gain information that can be used gain local access the victim system. An attacker can monitor keystrokes on the X server of a user who has left their X display open. A log of keystrokes is useful to an attacker because it might contain confidential information, or information that can be used to gain access to the system such as the username and password of the user being monitored.

Simulation Details

The Xsnoop program used in the simulation was adapted from C code originally posted to the Bugtraq mailing list. The Xsnoop program runs locally on the attacker's machine. The program connects to the victim's X Server and requests to be notified of all X KeyPress Events. If the attacker has permission to make this request (as would happen if the victim had typed "xhost +", or if the victim's X Server is configured to allow connections from all hosts by default) the Xsnoop program will be sent all KeyPress events that occur on the victim X Server. The Xsnoop program then uses the KeyPress events to provide the attacker with a view of all the keystrokes the victim. Because this attack required interaction with the X Server, it was always run by a human actor.

Attack Signature

An network based intrusion detection system can identify the Xsnoop attack by parsing the X protocol information in packets destined for remote X clients and noting that X KeyPress events are being transmitted to a remote machine. Because the Xsnoop attack results from bad security policy (leaving an X Server unsecured) and not simply a bug,

the presence of these packets alone does not signify that an attack is taking place. The intrusion detection system must know whether the security policy of the machine being monitored allows unauthenticated X connections from anywhere.

Chapter 9

Probes

In recent years, a growing number of programs have been distributed that can automatically scan a network of computers to gather information or find known vulnerabilities [27]. These network probes are quite useful to an attacker who is staging a future attack. An attacker with a map of which machines and services are available on a network can use this information to look for weak points. Some of these scanning tools (satan, saint, mscan) enable even a very unskilled attacker to very quickly check hundreds or thousands of machines on a network for known vulnerabilities. Figure 9-1 provides a summary of the probes that are discussed in the remainder of this chapter. The

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Ipsweep	ICMP	All	Abuse of Feature	Short	Finds active machines
Mscan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Nmap	many	All	Abuse of Feature	Short	Finds active ports on a machine
Saint	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Satan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities

Figure 9-1: Summary of Probes

following sections describe in detail each of the probes that was used in the 1998 DARPA intrusion detection evaluation.

9.1 Ipsweep

R-a-Probe(Machines)

Background

An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines.

Simulation Details

There are many methods an attacker can use to perform an Ipsweep attack. The most common method—and the method used within the simulation—is to send ICMP Ping packets to every possible address within a subnet and wait to see which machines respond. The Ipsweep probes in the simulation were not stealthy—the sweeps were performed linearly, quickly and from a single source.

Attack Signature

An intrusion detection system looking for the simple Ipsweep used in the simulation can look for many Ping packets, destined for every possible machine on a network, all coming from the same source.

9.2 Mscan

R-a-Probe(Known Vulnerabilities)

Description

Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities [20].

Simulation Details

The Mscan program used in the simulation was compiled from source code found at [57]. Mscan was easy to run and has several command line options for specifying the number of machines to scan and which vulnerabilities to look for. Within the simulation, mscan was used to scan the entire eyrie.af.mil domain for the following vulnerabilities: statd, imap, pop, IRIX machines that have accounts with no passwords, bind, various cgi-bin vulnerabilities, NFS, and open X servers.

Attack Signature

The signature of this attack will vary depending on which vulnerabilities are being scanned for and how many machines are being scanned. In general, an intrusion detection system can find an Mscan attack by looking for connections from a single outside machine to the ports listed above on one or more machines within a short period of time.

9.3 Nmap

R-a-Probe(Services)

Description

Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscans—options include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning [45]. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order.

Simulation Details

At the time of the evaluation, Nmap was the most complete publicly available scanning tool. During the simulation, Nmap was used to perform portscans on between one and ten computers using SYN scanning, FIN scanning, and UDP scanning of victim machines. Both sequential and random scans were performed in the simulation, and the timeout between packets was varied to be anywhere from one second to six minutes. The number of ports scanned on each machine was varied between three and one thousand.

Attack Signature

The signature of a portscan using the Nmap tool varies widely depending on the mode of operation selected. Despite this variance of modes, all portscans share some common features. A portscan can be recognized by noting that network packets (whether via TCP or UDP, or via only FIN packets or only SYN packets) have been sent to several (or more) ports on a victim or group of victims within some window of time. Two factors complicate the identification of portscans. First, a portscan can happen very slowly. An attacker who is patient could probe one port per day. Current intrusion detection systems do not keep enough state to recognize a portscan happening over such a long period of

time. With the amount of network traffic sent over a typical network, keeping enough active state within the intrusion detection system to recognize one connection per day for one hundred days as a one hundred day long portscan is simply not practical. Second, the connections don't necessarily all have to come from the same host. A group can perform a coordinated scan with each member scanning only a subset of machines or ports. By combining these methods, a group could perform a "low/slow" portscan that would be very hard to recognize.

9.4 Saint

R-a-Probe(Known Vulnerabilities)

Description

SAINT is the Security Administrator's Integrated Network Tool. In its simplest mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, statd, and other services. The information gathered includes the presence of various network information services as well as potential security flaws. These flaws include incorrectly setup or configured network services, well-known bugs in system or network utilities, and poor policy decisions. Although SAINT is not intended for use as an attack tool, it does provide security information that is quite useful to an attacker [58].

Simulation Details

SAINT is distributed as a collection of perl and C programs and is known to run on Solaris, Linux, and Irix systems. Within the simulation, the Saint program was run from a Linux traffic generator and was used to probe several victim machines for vulnerabilities. SAINT's behavior is controlled by a configuration file which allows the user to specify several parameters. The most important parameters are the list of machines to scan, and how heavily to scan these machine (light, normal, or heavy). In light mode, SAINT will probe the victim for dns and rpc vulnerabilities and will look for unsecured NFS mount points. In normal mode SAINT will also check for vulnerabilities in fingerd, rusersd, and bootd, and will perform a portscan on several tcp (70, 80,ftp, telnet, smtp, nntp, uucp) and udp (dns, 177) ports. Heavy mode is the same as normal mode except that many more ports are scanned.

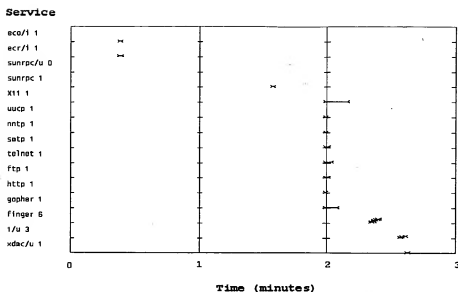


Figure 9-2: Plot of Connections During a Medium Level Saint Scan

Attack Signature

A Saint scan of a network leaves a distinct signature that will vary depending on the level of scanning being performed. The Saint program performs each scan in a nearly deterministic fashion. To identify a Saint scan, an intrusion detection system needs to be able to recognize the distinct set of network traffic the scan creates. Figure 9-2 is a plot that provides a graphical view of this signature. The horizontal axis of this plot represents time in minutes, and the different services probed are presented along the vertical axis. The names of the services are shown on the left side of this plot, and connections for each service are plotted within each named region. The numbers after the service names are the number of separate tcp connections or of udp or icmp packets. Names ending in “/i” indicate that packets use the icmp protocol and names ending in “/u” indicate that packets use the udp protocol. All other services use the tcp protocol.

Each line segment represents a connection to a service. This plot shows the unique signature of a medium level Saint scan. Because this signature does not change significantly across multiple instantiations of the Saint attack, an intrusion detection system that has been trained to recognize the pattern of connections shown in Figure 9-2 will probably detect other Saint attacks.

9.5 Satan

R-a-Probe(Known Vulnerabilities)

Description

SATAN is an early predecessor of the SAINT scanning program described in the last section. While SAINT and SATAN are quite similar in purpose and design, the particular vulnerabilities that each tool checks for are slightly different [24].

Simulation Details

Like SAINT, SATAN is distributed as a collection of perl and C programs that can be run either from within a web browser or from the UNIX command prompt. SATAN supports three levels of scanning: light, normal, and heavy. The vulnerabilities that SATAN checks for in heavy mode are:

- NFS export to unprivileged programs
- NFS export via portmapper
- NIS password file access
- REXD access
- tftp file access
- remote shell access
- unrestricted NFS export
- unrestricted X Server access
- writeable ftp home directory
- several Sendmail vulnerabilities
- several ftp vulnerabilities

Scans in light and normal mode simply check for smaller subsets of these vulnerabilities.

Attack Signature

A SATAN scan of a network can be recognized by the consistent pattern of network traffic the program creates. The checks for the vulnerabilities listed above are always performed in the same order.

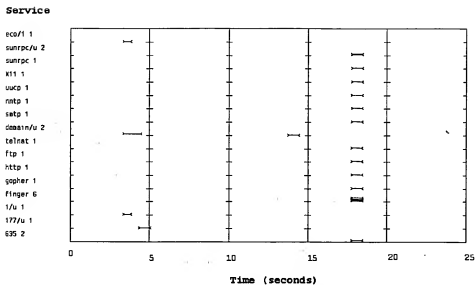


Figure 9-3: Plot of Connections During a Medium Level Satan Scan

Figure 9-3 shows a plot of the services probed during an example medium level Satan scan. The horizontal axis of this plot represents time in seconds and the various services that are probed are presented on the vertical axis. Each line segment in the plot represents a single connection to a service. Every medium level Satan scan will have a signature very similar to that shown in Figure 9-3.

Chapter 10

Realistic Intrusion Scenarios

An attack against a computer system can consist of several phases. An attacker might run several probes against a system looking for weaknesses, then use information gained from these probes to exploit some security vulnerability and gain access to the system.

Later an attacker can use this access to install a backdoor in the system, or plant a “logic bomb” in some program that causes damage at some point in the future. The actions of a single intruder might be spread out over the course of weeks or even months. Some attackers make no attempt to hide their intrusions from systems administrators while others are very careful to cover their tracks. In order to accurately test an intrusion detection system’s effectiveness in finding real attackers, it was necessary to not only develop a database of realistic exploits and probes, but to combine these exploits and probes with probable attacker actions and realistic goals to generate complete attack scenarios. This chapter describes how these various parameters of attacker behavior were varied to create a realistic simulation of the wide variety of attacker behavior that is manifested by actual intrusions.

10.1 Attack Scenarios

Most of the attacks that were included in the evaluation consist of a single session, or a few sessions that all occur within some short period of time. In the real world, an

attacker often has a goal in mind, and sometimes this goal cannot be achieved in a single session. Several complex scenarios were added to the collected data in an attempt to create a better simulation of real attacker behavior. Each scenario consists of a planned sequence of sessions (sometimes over the course of a week or more) that represent the actions of a single individual in pursuit of a goal. The following subsections describe each scenario in detail.

10.1.1 Cracker

The default scenario that occurred in 95% of sessions is that a curious cracker was trying to gain access to a machine just to prove that it could be done. Usually these crackers are simply trying to break into as many machines as they can, and may install a backdoor or download the password file in order to guarantee that they can access the machine again. Crackers are represented as individuals of different skill levels, some perform all of their actions in the clear, while others are aware that an intrusion detection system is present and take actions to avoid detection.

10.1.2 Spy

The spy is an information collector who comes back to a compromised machine several times to collect information. A spy might be looking for confidential data files or reading user's personal mail. A spy will take steps to minimize the possibility of detection.

10.1.3 Rootkit

The rootkit scenario could be viewed as an extension of the default Cracker scenario.

A rootkit is a collection of programs that are intended to help a hacker maintain access to a machine once it has been compromised. A typical rootkit consists of a sniffer, versions of login, su, and other programs with backdoors which allow for access, and new versions of ps, netstat, and ls that hide the fact that a sniffer is running and hide files in certain directories. Once the rootkit has been installed, the attacker comes back several times to download the sniffer logs.

10.1.4 Http Tunnel

The Http Tunnel scenario was originally developed as a method of defeating a firewall and for continuing to access a system while minimizing the chance of detection. The http tunnelling tools used in this scenario were developed specifically for use in this evaluation. Figure 10-1 provides an overview of this scenario. Assuming that a hacker is able to penetrate a firewall once (perhaps by sending an email with the executable content of the client in it or by dialing into a modem) a server program can be installed that masquerades as a normal user browsing web pages. Once this server has been installed, the hacker can issue requests to execute commands or transfer files with interaction happening in web traffic between the server and the hacker. For the simulation, the data was exchanged through cookies that rode along with a web request. This general method is very flexible however, and the data could have been tunneled using any cryptographic or steganographic technique. Tunneling through http was chosen with hopes that the large amount of http traffic most networks see in a typical day would obscure the actions of the attacker.

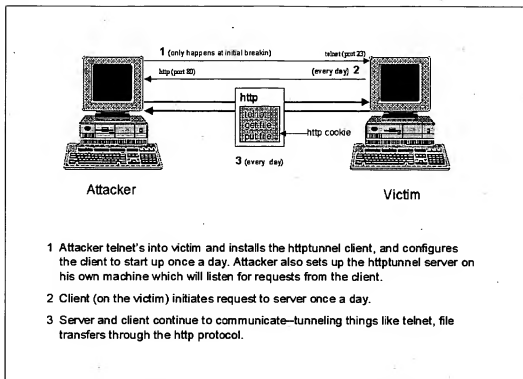


Figure 10-1: Attacker Uses Http to Tunnel Through a Firewall

10.1.5 SNMP Monitoring

An attacker who has guessed the SNMP community password of a router will then be able to monitor the traffic levels on that router, and may be able to issue commands to the router to change default routes or allow connections from a previously forbidden host or network.

10.1.6 Multihop

Some intrusion detection systems monitor traffic immediately outside of a router and only see traffic going into or coming out of that network. The multihop scenario was designed to test whether these systems could find attacks where an attacker first breaks into one inside machine, and then uses this inside machine for further attacks on the rest

of the machines on the network. This would be a very effective means of launching a Denial of Service attack undetected, as an intrusion detection system which sits just outside the network being monitored would not see a Denial of Service attack that originated from the internal network.

10.1.7 Disgruntled/Malicious User

These sessions simulate an attacker who is not interested in collecting information from a system or gaining access to a system, but is simply interested in doing damage. This is a common threat in operational computer networks [32]. Within the simulation, one malicious user re-formatted the primary disk partition of a victim machine.

Chapter 11

Stealthiness and Actions

In addition to varying the methods and intentions of the simulated attackers, attention was given to the extent to which attackers tried to hide their actions from either an individual who is monitoring the system, or an intrusion detection system. There are several ways that attackers can reduce their chances of being detected by the administrator of a network. Skilled attackers might try to cover their tracks by editing system logs or resetting the modification date on files that they replaced or modified. These actions are generally intended to reduce the chance of detection by a human administrator. Attackers may also be aware that an intrusion detection system is monitoring a network, and may try to hide from the intrusion detection system as well. Methods for being stealthy vary depending on the type of attack.

11.1 Avoiding Detection of Denial of Service (R-Deny)

Denial of Service attacks are difficult to make stealthy. One method an attacker can use to hide a denial of service attack is to gain the cooperation of a large group and break up the attack so pieces of it are coming from several different sources. Another method an attacker can use is to send thousands of packets with different spoofed source-addresses. Sending these spoofed packets will not make identifying the attack any harder, but they will make it more difficult to track down and stop the attacker because the victim has no

way of knowing which of the thousands of addresses the actual attack is coming from. Both of these methods do not make the attack any harder to detect, but simply reduce the chances that the attacker will be caught. No stealthy denial of service attacks were included in the 1998 DARPA intrusion detection evaluation.

11.2 Avoiding Detection of Probes (R-Probe)

Several methods can be used either to hide the fact that a probe is occurring, or obscure the identity of the party who is performing the probe. The following paragraphs describe methods of increasing stealth that were used in the probes included in the 1998 DARPA intrusion detection evaluation.

Scan Slowly and Randomly: If an attacker wants to hide the fact a probe is occurring, the probe can be configured to occur slowly and probe ports or machines in a non-linear order. An intrusion detection system will have a very hard time identifying one stray connection per hour to a random port as a port sweep initiated by an attacker.

Probe With Half-Open or Other Unlogged Connections: Another method of scanning stealthily is to probe with half-open connections. A connection for which the three-way TCP handshake is never completed will not be logged by the operating system. There are several tools available that will perform this type of half-open (FIN) scanning of a network.

Use an Intermediate Machine to Obscure the Real Source of the Scan: One way attackers can hide their identity is to use an ftp bounce probe. Some ftp servers will allow anyone to tell them to send data to a particular port on a particular machine. An attacker can look at the response the ftp server gives from such a request and ascertain whether that port is listening on the victim machine. The portscan will appear to be

coming from an anonymous ftp server, and this simple step may be enough to assure that the party who is really doing the scanning is never identified.

11.3 Avoiding Detection of User to Root (L-?-S) Attacks

There are many ways that User to Root attacks can be made stealthy. The following paragraphs discuss each of the methods that were used to make a number of the User to Root attacks in the 1998 DARPA intrusion detection evaluation stealthy.

Keyword Hiding: Some intrusion detection systems that attempt to detect illegal User to Root transitions rely on keyword spotting to detect intruders. For example, if the system observes the text of the C source code for the publicly available Eject exploit in a telnet or rlogin session, it will flag this session as being suspicious. By uuencoding or Mime encoding the text of the code before sending it over the network connection, an attacker would avoid detection by such a system.

Output Hiding: It is possible to identify attacks by looking at the output that is displayed on the terminal when the exploit is run. An attacker can avoid detection via this mechanism by sending all the output of commands that are run to a file and encoding (again with some method like ROT13, uuencode, or gzip) the file before displaying or transferring it.

Command Hiding: A system might also look for an attacker to run some command that only the superuser should be able to run, such as displaying the contents of the shadow password file. The invocation of a command that the attacker wishes to hide from someone who is looking for certain suspicious commands or actions can be obfuscated by using glob constructs and character replacement. Instead of typing the command "cat /etc/passwd", the attacker can issue the command "[r,s,t,b]?[l,w,n,m]/[c,d]?t

`/t{c,d,e}/*a?s*`". When the shell tries to interpret this input string it will do replacement of the glob characters and find that the only valid match for this string is `"/bin/cat /etc/passwd"`.

Delayed Attack: An attacker can also avoid detection (or at least reduce the chances of identification), by separating the time of exploit from the initial time of access. This can be accomplished by submitting a job to the "at" or "cron" daemon which will run the attack at some later time. An attacker can log in as a normal user (which would not be noticed by an intrusion detection system) and submit a shell script to the "at" daemon which would execute any actions that the attacker desired three weeks (or six months, or five years!) after the attacker initially logged into the system. Figure 11-1 provides a graphical illustration of a delayed attack scenario. This scenario is similar to one that was used in the evaluation. First, at time segment 1 in the figure, the attacker uses a sniffed password to log into the victim machine as a normal user. This action will not be noticed by an intrusion detection system. Before logging out, the attacker submits a job to the cron or at daemon that performs a User to Root exploit and some malicious actions at some point in the future. At time segment 2 in the figure, the submitted script is run and the exploit occurs. Later, at time segment 3 the attacker can come back and log into the machine as a normal user to collect the output from the commands run during the second segment, again without being noticed. The only step in this process that is easily detectable by an intrusion detection system is step 2 when the actual exploit occurs. However, during this time segment the attacker is not logged in to the system. In order to successfully identify this attack, an intrusion

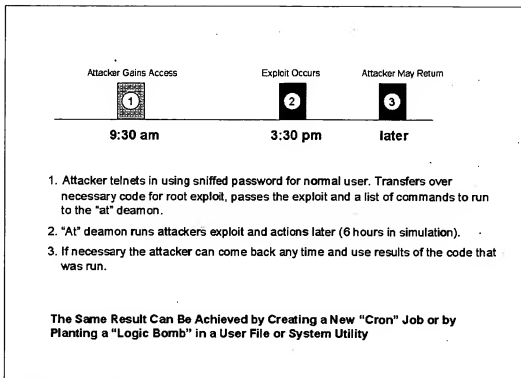


Figure 11-1: Attacker Can Use "at" to Temporally Dissociate the Time of Access and Time of Attack

detection system would need to correlate the earlier session to the malicious actions that occur when the attacker is not logged in.

Figure 11-2 is a transcript from an actual stealthy eject attack used in the simulation. This attack session contains the keyword hiding, output hiding, and command hiding techniques discussed above. Lines 10 through 16 contain the output of the attacker transferring the exploit files to the victim system. The files have been uuencoded to help avoid detection of the attack by a system that is looking for keywords that might be located in the text of the exploit files.

```

1 UNIX(r) System V Release 4.0 (pascal)
2
3 login: alie
4 Password:
5 Last login: Thu Jul 30 09:15:04 from 135.8.60.182
6 Sun Microsystems Inc. SunOS 5.5 Generic November 1995
7 pascal>
8 pascal> which gcc
9 /bin/gcc

```

```

10 pascal> uudecode
11 begin 644 /tmp/16256.c
12 M("@"@("V"B"@("I3;VQA<FES(#[N-2XQ("T@=&AI<R!E>1L;VET960@=-V%$
13 M(&-O;7!1;83D(&]N(%-O;&%R;7,R+C0@86YD(1E<W1E9"IO;@H@("@"@,BXU
14
15 end
16 <FILE TRANSFER TRUCATED FOR EXAMPLE>

```

**Keyword
Hiding**

```

17 pascal>
18
19 pascal> /bin/gcc -o /tmp/162562 /tmp/16256.c
20 pascal> which gcc
21 /bin/gcc
22 pascal> /bin/gcc -o /tmp/162563 /tmp/16256.c
23 pascal> uudecode
24 begin 644 /tmp/16256.c
25 M(VEN8VQU9&4@/SN-7-T9"YH@|V,VED"FU&6XH6YT(&%R9VJL(&-H87@
26 M*F(R9W9;72*->PH@('E'=6ED*#L"D"|"B"@97AE8VPH(B)B:6XO=&-S
27 O-("L(GIC<V@B+P#1P1]"@''
28
29 end
30 pascal>
31
32 pascal> /tmp/162562
33 Jumping to address 0xefff6a0 B[364] E[400] SO[704]
34 #

```

```

35 #ls |/u7n/7c?/b7n/g?ip |/b7n/u7e?c?d? gfd
36 begin 644 gfd
37 B7XL( (%@.P34 RM)+2XQY"HID7K)8-J("P"DGTI%
38
39 end
40 # /u7n/b7n/i? |/u7n/7c?/b7n/g?ip |/b7n/u7e?c?d? asd
41 begin 644 asd
42 MXL(%L:P34 RO-3+SU,C0TTS,C,R4S555@<@T-##1*BU+BC454DN!?.
43 /HOS$SDTN CL?1@K
44
45 end
46 # /b?/c?/et?/p?/w? |/u7n/7c?/b7n/g?ip |/b7n/u7e?c?d? hghhgf
47 begin 644 hghhgf
48 MXL(%L:P34 SZ?7 <QW7G[(&(!@A 8@!"/ZP10%*R1.*-) C+UH( ""%)
49 <FILE TRANSFER TRUCATED FOR EXAMPLE>
50 1 "C"SO%+vU$3,-9$

```

**Command
Hiding and
Output Hiding**

```

51
52 end
53 # exit
54 pascal> logout

```

Figure 11-2: Transcript of a Stealthy Eject Attack

In lines 34 through 54 the attacker has already gained root access and is now performing some malicious actions. The attacker uses both command hiding with glob constructs and output hiding using both uuencode and gzip. For example, on line 46 the

attacker types “# /bin/?c?n /etc/?p?s?w? | /u?r/?c?l/?b?n/g?ip | /b?n/u?e?c?d? hghhgf”, which looks like garbage, but is interpreted by the shell as “# /bin/cat /etc/passwd | /usr/local/bin/gzip | /bin/uencode hghhgf”. Each of these techniques makes the attack harder to find for an intrusion detection system that is depending on recognition of key words for attack detection.

The User to Root attacks that were inserted into the data for the 1998 DARPA intrusion detection evaluation contained a mix of both stealthy and clear User to Root Attacks. For User to Root attacks in the four weeks of test data, 50 percent were performed in the clear with no actions, 25 percent were performed in the clear with actions, and 25 were performed stealthily with stealthy actions.

The stealthy methods presented in this section represent only a small subset of the possible steps an attacker could take in an attempt to make a User to Root attack less noticeable. Future evaluations will need to explore these additional methods more completely. As each generation of intrusion detection system learns how to defeat new stealthy methods, attackers will likely develop newer and better stealthy techniques.

11.4 Avoiding Detection of Remote to Local (R-?-L) Attacks

Many Remote to Local attacks have a unique signature that is hard to alter. In a dictionary-based password guessing attack the user simply must connect to a service many times and try different passwords. If an attacker wants to gain access to a machine by remotely overflowing the imap service they have to transmit the code that overflows the buffer across the network in the clear. For these reasons, there are fewer options for making Remote to Local attacks stealthy than there are for User to Root attacks. However, once an attacker has gained access and is interacting with a shell, all the same methods that were outlined in the User to Root section above can be used to hide further

actions the attacker might perform. None of the Remote to Local User attacks in the 1998 DARPA intrusion detection evaluation were designed to be stealthy.

11.5 Actions

Many of the attacks included within the data collected for the 1998 DARPA intrusion detection evaluation include some set of actions that were not directly related to the task of either gaining access or denying service. In some cases these actions were performed as part of a scenario where an attacker had a specific goal in mind. But, in many cases even attacks that were not developed with a specific scenario in mind contain a few actions that an attacker performs after gaining access. These actions were chosen from a set that was identified by looking at transcripts of actual intrusions and by considering the goals that an attacker might have. The following are descriptions of the actions that were performed during the simulation:

- **id** An attacker uses the "id" program to check whether an exploit obtains root access
- **cat /etc/passwd, cat /etc/shadow** Once the attacker has collected the contents of the password file the contents can be decrypted offline.
- **netstat** Netstat provides an attacker with a list of the network services a host offers.
- **mount, showmount** An attacker runs the mount or showmount command to look for NFS mounted drives that might provide access to another machine or additional data.
- **cat /etc/hosts.equiv** The "hosts.equiv" file contains a list of trusted machines. By adding new machines to this list of trusted hosts an attacker can assure later access.

- **creation of a suid root shell** By making a copy of /bin/sh or some other shell and setting its permissions to make the executable suid root the attacker can guarantee that they will later be able to obtain root access.
- **w, who** An attacker who is trying to avoid detection can use these commands to see who else is logged in to the system.
- **editing .rhosts** The .rhosts file is similar to /etc/hosts.equiv in that it specifies a list of trusted hosts. If an attacker can add the string "+ +" to /root/.rhosts the system will allow anyone to perform remote commands (rsh, rlogin, rcp, etc.) as root on that system.
- **uname** The uname command gives an attacker information about the hardware and operating system of the victim host.
- **starting up new services** Attacker can start a new listening port for later access to the machine. They can start up the tftp service that will later allow them to read any world-readable file on the system without authenticating themselves. They can bind a second listening telnet daemon to some port other than the normal port (port 23) to get around a firewall. Or, attackers can set up the netcat program to listen on a port and give them a shell, or provide them with some piece of information whenever someone connects to that port and supplies a "magic word". More examples are available in [31].

Chapter 12

Attack Planning and Data Collection

Nine weeks of data were collected during the 1998 DARPA intrusion detection evaluation. This data consisted of normal traffic as well as inserted attacks. Of these nine weeks of data, the first seven weeks are considered to be the “training” dataset and the last two weeks are the “testing” dataset. After the first seven weeks of data was collected, the data and an answer key listing the time, name, and affected machines for each attack was distributed to participants. The participants then trained their intrusion detection systems on this data. After the training phase was complete, two additional weeks of data was distributed. Participants attempted to find attacks in this two weeks of data and returned a list of found attacks to Lincoln Laboratory. These lists of found attacks were then scored to determine the participant’s success rate and false alarm rate across various categories of attacks. The remainder of this chapter discusses the method used to select attacks and keep track of them, shows the actual numbers of attacks present in the 1998 evaluation, and describes the process for validating that an attack occurred.

12.1 Planning and Keeping Track of Attacks

A complete schedule of all attacks for the nine week period was developed by hand prior to the beginning of the data collection process. The number of instances of each of the 32 attack types was kept roughly equal. Once the schedule was created, the schedule was

stored in electronic form and was used to create the "exs" scripts that were used to run each attack. A collection of CGI scripts was written to allow the schedule to be conveniently viewed in HTML format. The complete attack schedule for the two weeks of testing data can be found in Appendix A of this document.

12.2 Numbers of Attacks

Table 12-1 shows the number of instances of each type of attack that was inserted in the 1998 DARPA intrusion detection evaluation. Each row represents a type of attack. These attack types have been grouped by their category: User to Root, Remote to Local User, Denial of Service, or Probe. The second major column lists the number of instances of the current attack type that were performed in the clear. Within this second column the data is further divided into subcolumns listing the number of attacks of each type in the training data (first seven weeks), the testing data (last two weeks) and the total over all nine weeks of data. The third column shows the number of instances of each attack type that were performed stealthily. Note that no Remote to Local User attacks or Denial of Service attacks were stealthy. Finally, the third column aggregates the sums from the previous two columns, and shows the total number of instances (both clear and stealthy) of attacks of each type. For example, there were 46 Eject attacks in the simulation. Of these 46 Eject attacks, 10 were stealthy and 36 were performed in the clear. Of the 36 clear Eject attack instances, 29 were in the training data, and 7 were in the testing data.

Name	Attacks Performed in the Clear			Stealthy Attacks			All Attacks (Clear + Stealthy)
	Training	Testing	Total	Training	Testing	Total	Total
User to Root	67	24	91	18	5	23	114
Eject	29	7	36	8	2	10	46
Fdformat	12	7	19	4	3	7	26
Fbconfig	11	2	13	6	0	6	19
Perl	15	1	16	0	0	0	16
Ps	0	4	4	0	0	0	4
Xterm	0	3	3	0	0	0	3
Remote to Local User	19	15	34	0	0	0	34
Dictionary	3	2	5	0	0	0	5
Ftp-Write	2	1	3	0	0	0	3
Guest	4	0	4	0	0	0	4
Imap	4	1	5	0	0	0	5
Named	0	3	3	0	0	0	3
Phf	6	1	7	0	0	0	7
Sendmail	0	3	3	0	0	0	3
Xlock	0	2	2	0	0	0	2
Xsnoop	0	2	2	0	0	0	2
Denial Of Service	65	34	99	0	0	0	99
Apache2	0	3	3	0	0	0	3
Back	4	2	6	0	0	0	6
Land	7	2	9	0	0	0	9
Mailbomb	0	1	1	0	0	0	1
Neptune	13	7	20	0	0	0	20
Process Table	0	3	3	0	0	0	2
Smurf	19	8	27	0	0	0	27
Syslog	4	2	6	0	0	0	6
Teardrop	18	4	22	0	0	0	22
UDPSpoof	0	2	2	0	0	0	2
Probe/Surveillance	50	14	64	0	0	0	64
IPSweep	22	3	25	0	0	0	25
Mscan	0	1	1	0	0	0	1
Nmap	12	6	18	5	3	8	18
Saint	0	2	2	0	0	0	2
Satan	16	2	18	0	0	0	18
Total for All Attacks	201	87	288	18	5	23	311

Table 12-1: Instances of Non-Scenario Attacks

12.3 Verification of Attack Success

A major unanticipated difficulty of the process of creating the evaluation dataset was verifying that the inserted attacks actually ran and had the desired effect. Because the simulation network was not designed with automation of this task in mind, all verification of attack success was performed by a human operator after the data had been collected.

The verification process involved looking at logs generated by the "exs" script for each attack after it had run. This process was not ideal. In a more carefully designed verification strategy, the success or failure of an attack instance would be determined automatically at run time by looking at the actual data collected for that attack (tcpdump data, and BSM logs). Without such an automated system in place, the task of verifying attack success turned out to be quite time consuming and prone to error. A system for automatically validating attack success is being developed for use in the 1999 version of the Lincoln Laboratory DARPA intrusion detection evaluation.

Chapter 13

Results and Future Work

13.1 Results of the 1998 Evaluation

The 1998 DARPA intrusion detection evaluation was extremely successful in providing the first comprehensive realistic evaluation of intrusion detection systems. The results were useful in focusing research and highlighting the current capabilities and recent advances of existing intrusion detection systems. The procedures used to create the data and generate attacks worked well and can easily be reused to generate more data for future evaluations. Researchers who participated in the evaluation were able to see the strong and weak points of their own approaches, and the whole intrusion detection research community gained a great deal of insight about the strengths and weaknesses of current efforts in the field as a whole.

Although full discussion of the detailed results of the evaluation are outside of the scope of this thesis, there were a few broad observations that will be useful in guiding future efforts to evaluate intrusion detection systems using simulated computer attacks.

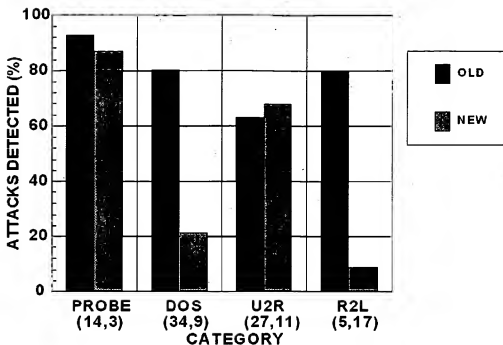


Figure 13-1: Current Intrusion Detection Systems Cannot Find New DoS and Remote to User Attacks

First, current intrusion detection research systems represent a dramatic improvement over older keyword spotting systems. The best combination of 1998 evaluation systems provides more than two orders of magnitude of reduction in false alarm rate with greatly improved detection accuracy. Second, current Intrusion Detection Systems can successfully identify older, known attacks with a low false alarm rate, but do not perform as well when identifying novel or new attacks. Figure 13-1 graphically shows this result by comparing the detection rates of the three best evaluated systems that used tcpdump data. The vertical axis of the chart represents the percentage of attacks detected when set to a threshold that allows for 10 false alarms/day. Results are broken up into the four attack categories (Probe, Denial of Service, User to Root, and Remote to Local) along the horizontal axis. This figure shows us that systems were able to

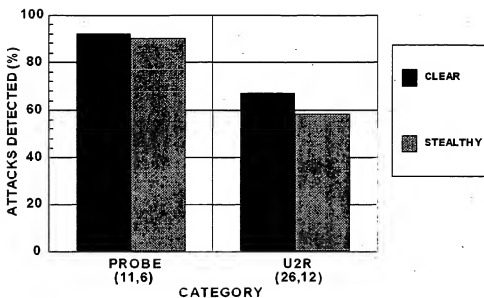


Figure 13-2: Clear vs. Stealthy Attack Detection Rates
(Top 3 Systems, TCPDump Data, 10 false alarms/day)

generalize well to detect new Probe and User to Root attacks, but almost completely missed new Denial of Service and Remote to Local attacks.

Finally, the stealthy approaches presented in Chapter 11 did not significantly degrade the performance of the evaluated systems. Figure 13-3 compares the detection rate (for the top three systems at a false alarm rate of 10/day) for clear and stealthy Probes and User to Root Attacks. The rate of detection is represented on the vertical axis and Results are broken up into the two attack categories that can be made stealthy (Probes and User to Root) along the horizontal axis. Detection rates for stealthy attacks were not significantly lower than for attacks performed in the clear. Two possible reasons for the relative ineffectiveness of the stealthy attacks are the fact that many stealthy attacks were included in the training dataset, and the fact that many of the stealthy approaches discussed in Chapter 11 were designed specifically to thwart systems

that rely on keyword spotting. More detailed descriptions of these results can be found at the Lincoln Laboratory Intrusion Detection Evaluation Website [42].

13.2 Future Work

13.2.1 Inclusion of Windows NT and Other Systems in the Evaluation

The 1998 Intrusion Detection evaluation focused on attacks targeted at UNIX servers, because at the current time these UNIX based machines are the most common server platform within the defense community. However, use of Windows NT servers is growing in both military and commercial environments, and attacks against Windows NT servers should be included in future evaluations. As the landscape of computer networks continues to evolve, the evaluation must evolve and adapt in order to maintain its relevance and usefulness.

13.2.2 Better Automation of Attack Planning and Verification

In the 1998 evaluation, all attack planning and verification was performed by hand. These tasks proved to be very time intensive and hindered the timely completion of the evaluation. In the future, this process should be automated. All that should be required of the human operator is to supply a database of attacks and their variants. Once this database has been collected, an automated system should be able to plan out a schedule, and generate the “exs” scripts for all of the attack instances in the evaluation. After the “exs” scripts have been run, automatic verification routines should be developed to look at the collected data and validate that the attacks occurred successfully. Developing these

automated tools will take a great deal of effort, but once they have been completed the process of conducting future evaluations will be greatly simplified.

13.2.3 Improved Stealthy Attacks

The results presented in section 13.1 show that the stealthy methods used in the 1998 DARPA evaluation were largely unsuccessful. Stealthy attacks were ineffective because

(1) Examples of stealthy attacks were provided in the training data and many intrusion detection systems were trained to find stealthy actions instead of the actual break-in, (2)

Only a small part of the development effort focussed on making attacks stealthy, and (3)

All of the user to root attacks created a root shell, which makes them relatively easy to detect. Future evaluation efforts should dedicate a large percentage of total attack development time to further developing and exploring new methods of achieving stealthiness.

Appendix A

Attack Schedule for Test Dataset

Day	Name	Time	Source	Dest	Type	Stealthy
Week 1						
Mon	portsweep	10:51:21	207.136.86.223	marx	probe	
Mon	pod	14:49:52	207.103.80.104	marx	dos	
Mon	teardrop	16:08:16	123.123.123.123	zeno	dos	
Mon	syslog	16:46:28	1.1.1.1	pascal	dos	
Tues	satan	08:31:46	153.107.252.61	marx	probe	
Tues	format	09:11:57	209.74.60.168	pascal	u2r	Y
Tues	apache2	10:11:30	207.181.92.211	marx	dos	
Tues	format2	15:44:55	194.27.251.21	pascal	u2r	
Tues	neptune	18:15:57	166.102.114.43	zeno	dos	
Tues	smurf	19:43:50	100subs	marx	dos	
Wed	httptunnel	09:32:05	197.182.91.233	pascal	scenario	
Wed	nmap	10:34:10	207.253.84.13	linux2	probe	
Wed	teardrop	16:30:05	123.123.123.123	zeno	dos	
Wed	smurf	20:07:38	199.174.194	marx	dos	
Wed	eject	21:52:01	153.10.8.174	pascal	u2r	Y
Thurs	pod	10:16:10	152.204.242.193	*	dos	
Thurs	nmap	10:23:19	207.253.84.13	linux2	probe	
Thurs	named	10:33:08	152.169.215.104	linux	r2l	
Thurs	eject	14:32:32	194.7.248.153	pascal	u2r	Y
Thurs	land	14:45:13	zeno	zeno	dos	
Thurs	smurf	15:10:48	10subs	linux3	dos	
Fri	snmpgetattack	08:03:06	207.230.54.203	loud	scenario	
Fri	teardrop	09:27:09	194.7.248.153	marx	dos	
Fri	format	10:06:37	194.7.248.153	pascal	u2r	Y
Fri	back	12:41:54	204.97.153.43	marx	dos	
Fri	neptune	14:41:28	9.9.9.9	pascal	dos	
Fri	processtable	19:20:35	Calvin	pascal	dos	
Fri	neptune	19:50:59	10.20.30.40	pascal	dos	
Week 2						
Mon	snmpgetattack	08:03:07	207.230.54.203	loud	scenario	
Mon	named	08:55:02	calvin	marx	r2l	
Mon	xlock	09:11:04	194.7.248.153	marx	r2l	
Mon	xlock2	09:27:27	194.7.248.153	pascal	R2l	
Mon	smurf	09:30:41	10subs	pascal	dos	
Mon	multihop	09:36:10	206.229.221.82	marx	scenario	
Mon	ipsweep	09:36:07	204.97.153.43	*	probe	
Mon	xsnop	09:38:58	194.27.251.21	pascal	r2l	
Mon	named	09:54:39	204.97.153.43	linux1	r2l	
Mon	smurf	10:34:11	all.attackers	marx	dos	
Mon	saint	11:01:23	calvin	pascal	probe	
Mon	ffb	11:15:16	197.218.177.69	pascal	u2r	
Mon	portsweep	14:01:26	202.247.224.89	pascal	probe	
Mon	pod	16:11:36	207.103.80.104	pascal	dos	
Mon	apache2	18:22:34	196.227.33.189	marx	dos	
Mon	format	20:46:48	195.73.151.50	pascal	u2r	
Mon	udpstorm	21:06:52	172.16.112.50	zeno	dos	
Mon	format2	23:10:31	202.49.244.10	Pascal	u2r	
Mon	smurf	23:16:43	152.169.215.202	Marx	dos	

			.77.162			
Mon	smurf	23:26:37	255.255.255.255 .et.al	Pascal	dos	
Tues	warezmaster	08:08:29	200.27.121.118	Pascal	scenario	
Tues	portsweep	08:27:28	135.8.60.182	Marx	dos	
Tues	multihop	09:00:19	206.229.221.82	Marx	scenario	
Tues	perlmagic	09:08:32	166.102.114.43	Marx	u2r	
Tues	xsnop	09:09:03	200.27.121.118	Pascal	r2l	
Tues	pod	10:33:08	209.30.70.14	Linux1	dos	
Tues	dict1	11:23:37	208.239.5.230	Marx	r2l	
Tues	format1	11:55:17	209.30.71.165	Pascal	u2r	Y
Tues	sendmail	12:14:12	204.97.153.43	Marx	r2l	
Tues	eject	13:20:35	195.115.218.108	Pascal	u2r	
Tues	satan	15:17:26	208.253.77.185	Zeno	probe	
Tues	mcan	17:36:25	207.75.239.115	*	probe	
Tues	processtable	18:04:07	calvin	Pascal	Dos	
Tues	smurf	21:03:36	209.1.12.*	Linux2*	dos	
Wed	snmpgetattack	08:03:06	207.230.54.203	Loud	scenario	
Wed	warezclient	08:15:06	all.attackers	Pascal	scenario	
Wed	ps	10:06:44	209.154.98.104	Pascal	u2r	
Wed	nmap	10:16:15	207.253.84.13	Linux2	probe	
Wed	dict1	10:24:39	152.169.215.104	Marx	r2l	
Wed	xterm	10:26:12	194.27.251.21	Linux1	u2r	
Wed	rootkit	10:33:27	153.10.8.174	Marx	scenario	
Wed	xterm	10:52:58	197.182.91.233	Linux1	r2l	
Wed	neptune	11:11:28	10.140.175.80	Zeno	dos	
Wed	udpstorm	12:40:23	pascal	Zeno	dos	
Wed	loadmodule	13:32:38	205.180.112.36	Zeno	u2r	
Wed	syslog	13:46:54	194.7.248.153	Pascal	dos	
Wed	imap	14:36:15	calvin	Marx	r2l	
Wed	back	15:46:55	209.117.157.183	Marx	dos	
Wed	ps	19:26:57	197.218.177.69	Pascal	u2r	
Wed	ipsweep	20:07:16	194.7.248.153	*	probe	
Wed	ffb	20:59:25	208.239.5.230	Pascal	u2r	
Thurs	snmpgetattack	08:03:06	207.230.54.203	Loud	Scenario	
Thurs	warezclient	08:15:05	all.attackers	Pascal	Scenario	
Thurs	multihop	09:00:09	206.229.221.82	Pascal	Scenario	
Thurs	eject	09:15:00	135.008.060.182	Pascal	u2r	
Thurs	portsweep	09:36:05	195.73.151.50	Pascal	Probe	Y
Thurs	nmap	09:41:52	207.253.84.13	Linux2	probe	
Thurs	sendmail	10:44:06	207.230.54.203	Marx	R2l	
Thurs	loadmodule	11:37:27	135.13.216.191	Zeno	U2r	
Thurs	eject	11:59:43	197.182.91.233	Pascal	U2r	
Thurs	httptunnel	12:30:03	pascal	Calvin	scenario	
Thurs	neptune	13:30:49	209.74.60.168	Zeno	Dos	
Thurs	eject2	13:42:55	197.182.91.233	Pascal	U2r	
Thurs	processtable	17:56:34	calvin	Pascal	Dos	
Thurs	neptune	20:15:14	135.13.216.191	Pascal	Dos	
Thurs	format	21:13:12	206.186.80.111	Pascal	U2r	
Thurs	mailbomb	21:14:46	204.233.47.21	Pascal	Dos	
Thurs	nmap	23:26:16	128.223.199.68	Zeno	Probe	
Fri	snmpgetattack	08:04:11	207.230.54.203	Loud	Scenario	
Fri	warezclient	08:16:11	all.attackers	Pascal	Scenario	
Fri	xterm	08:43:22	calvin, 209.167.99.71	Linux1	u2r	
Fri	ftp-write	08:49:26	194.27.251.21	Pascal	R2l	
Fri	ps	08:53:05	207.136.86.223	Pascal	U2r	
Fri	httptunnel1	09:01:08	pascal	Mars	scenario	
Fri	teardrop	09:33:40	222.222.222.222	Marx	Dos	
Fri	saint	10:20:15	197.218.177.69	*	probe	

Fri	neptune	10:30:11	18.28.38.48	Pascal	Dos	
Fri	ps2	12:27:21	202.49.244.10	Pascal	u2r	
Fri	eject	12:48:24	153.107.252.61	Pascal	u2r	
Fri	land	13:27:39	Zeno	Zeno	u2r	
Fri	sendmail	16:22:02	194.7.248.153	Marx	R2l	
Fri	ipsweep	16:46:15	135.13.216.191	*	Probe	
Fri	pod	18:16:42	196.227.33.189	linux1	Dos	
Fri	phf	20:18:38	197.182.91.233	Marx	r2l	
Fri	apache2	21:04:50	202.72.1.77	Marx	dos	

References

- [1] Anderson, D., T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. "Safeguard Final Report: Detecting Unusual Program Behavior Using the NIDES Statistical Component," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, December 1993.
- [2] Anonymous. *Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network*, Chapter 15, pp. 359-362. Sams.net, 201 West 103rd Street, Indianapolis, IN, 46290. 1997.
- [3] Bishop, M., S. Cheung, et al. "The Threat from the Net", IEEE Spectrum, 38(8). 1997.
- [4] Bugtraq Archives (e-mail regarding Apache vulnerability). http://www.geek-girl.com/bugtraq/1998_3/0442.html. August 7, 1998.
- [5] Bugtraq Archives. http://geek-girl.com/bugtraq/1997_4/0283.html. November 13, 1999.
- [6] Bugtraq Archives. http://www.geek-girl.com/bugtraq/1999_1/0852.html. February 19, 1999.
- [7] CERT Advisory CA-93.10. http://www.cert.org/ftp/cert_advisories/CA-93%3a10.anonymous.FTP.activity. July 14, 1993.
- [8] CERT Advisory CA-93.18,CA-95:12. http://www.cert.org/ftp/cert_advisories/CA-95:12.sun.loadmodule.vul. September 19, 1997.
- [9] CERT Advisory CA-95.09. http://www.cert.org/ftp/cert_advisories/CA-95%3a09.Solaris-ps.vul. August 20, 1995.
- [10] CERT Advisory CA-96.01. http://www.cert.org/ftp/cert_advisories/CA-96.01.UDP_service_denial. February 8, 1996.
- [11] CERT Advisory CA-96.06. http://www.cert.org/ftp/cert_advisories/CA-96.06.cgi_example_code. March 20, 1996.
- [12] CERT Advisory CA-96.12. http://www.cert.org/ftp/cert_advisories/CA-96.12.suidperl_vul. June 26, 1996.
- [13] CERT Advisory CA-96.21. http://www.cert.org/ftp/cert_advisories/CA-96.21.tcp_syn_flooding. September 19, 1996.
- [14] CERT Advisory CA-96.26. http://www.cert.org/ftp/cert_advisories/CA-96.26.ping. December 16, 1996.
- [15] CERT Advisory CA-97.05. http://www.cert.org/ftp/cert_advisories/CA-97.05.sendmail. January 28, 1997.
- [16] CERT Advisory CA-97.09. http://www.cert.org/ftp/cert_advisories/CA-97.09.imap_pop. April 7, 1997.
- [17] CERT Advisory CA-97.28. http://www.cert.org/ftp/cert_advisories/CA-97.28.Teardrop_Land. December 16, 1997.
- [18] CERT Advisory CA-98.01. http://www.cert.org/ftp/cert_advisories/CA-98.01.smurf. January 5, 1998.

- [19] CERT Advisory CA-98.05. http://www.cert.org/ftp/cert_advisories/CA-98.05.bind_problems. April 8, 1998.
- [20] CERT Incident Note. http://www.cert.org/incident_notes/IN-98.02.html. July 2, 1998.
- [21] CERT Vulnerability Note VN-98.01. http://www.cert.org/vul_notes/VN-98.01.XFree86.html. May 3, 1998.
- [22] Computer Emergency Response Team Website. <http://www.cert.org>.
- [23] Cisco Systems, Inc. "NetRanger Intrusion Detection System Technical Overview," http://www.cisco.com/warp/public/778/security/netranger/ntran_tc.htm.
- [24] COAST FTP Site. <ftp://coast.cs.purdue.edu/pub/tools/unix/satan/>. 1998
- [25] Cunningham, R. K., R. P. Lippmann, D. J. Fried, S.L. Garfinkel, I. Graf, K. R. Kendall, S.E. Webster, D. Wyschogrod, and M. A. Zissman, (1999) "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation," In Proceedings ID '99, Third Conference and Workshop on Intrusion Detection and Response, San Diego, CA: SANS Institute.
- [26] S. Durst, T. Champion. Packet Address Swapping for Network Simulation. Patent application, Air Force Research Laboratory. March 1999.
- [27] Simon Garfinkel and Gene Spafford. *Practical Unix & Internet Security*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, 2nd edition, April 1996.
- [28] Heberlein, T. "Network Security Monitor (NSM) – Final Report", U.C. Davis: February 1995, <http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf>.
- [29] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the 1990 Symposium on Research in Security and Privacy*, pages 296-304, Oakland, CA, May 1990. IEEE.
- [30] Heberlein, T.L., G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. "A Network Security Monitor", in 1990 IEEE Symposium on Research in Security and Privacy. pp. 296-304.
- [31] *hobbit* hobbit@avian.org. Netcat README file. <http://www.l0pht.com/~weld/netcat/readme.html>. 1998.
- [32] D. Icove, K. Seger, W. VonStorch. *Computer Crime: A Crimefighter's Handbook*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, August 1995.
- [33] Koral Ilgun. USTAT : A real-time intrusion detection system for UNIX. Master's thesis, University of California Santa Barbara, November 1992.
- [34] Impack binaries on Rootshell.com. <http://www.rootshell.com/archive-j457nxiq3gq59dw/199804/impack103.tar.gz.html>. April 13, 1998.
- [35] Internet Security Systems X-Force. <http://www.iss.net>.
- [36] R. Jagannathan, Teresa Lunt, Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalai, Hal Havitz, Peter Neumann, Ann Tamaru, and Alfonso Valdes. System design document: Next-generation intrusion detection expert system (NIDES). Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, March 1993.

- [37] Javitz, H.S. and A. Valdes, "The NIDES Statistical Component Description and Justification," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, March 1994.
- [38] Kemmerer, Rachard A. "NSTAT: A Model-based Real-time Network Intrusion Detection System," Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, <http://www.cs.ucsb.edu/TRs/TRCS97-18.html>
- [39] Ko, C., M. Ruschitzka, and K. Levitt. "Execution Monitoring of Security-Critical Programs in a Distributed System: A Specifications-Based Approach," In Proceedings 1997 IEEE Symposium on Security and Privacy, pp. 134-144, Oakland, CA: IEEE Computer Society Press.
- [40] Lawrence Berkeley National Laboratory, Network Research Group Homepage. <http://www-nrg.ee.lbl.gov/>. May 1999.
- [41] Lawrence Livermore National Laboratory. "Network Intrusion Detector (NID) Overview," Computer Security Technology Center, <http://ciac.llnl.gov/cste/nid/intro.html>.
- [42] Lincoln Laboratory ID Evaluation Website, MIT, <http://www.ll.mit.edu/IST/ideval/index.html>. 1999.
- [43] Lippmann, R.P., Cunningham, R.K., Webster, S.E., Graf, I., Fried, D. Using Bottleneck Verification to Find Novel New Computer Attacks with a Low False Alarm Rate. Unpublished Technical Report. 1999.
- [44] Lunt, T.F. "Automated Audit Trail Analysis and Intrusion Detection: A Survey", in Proceedings 11th National Computer Security Conference., pp. 65-73. 1988.
- [45] NMAP Homepage. <http://www.insecure.org/nmap/index.html>. 1998
- [46] Parras, Phillip A. and Peter G. Neumann. "EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances," In Proceedings 20th National Information Systems Security Conference, Oct 7, 1997.
- [47] Paxon, Vern. "Bro: A System for Detecting Network Intruders in Real-Time," In Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998, <http://www.aciri.org/vern/paper.html>.
- [48] Nicholas Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, Ronald Olsson. A Methodology for Testing Intrusion Detection Systems. Technical report, University of California, Davis, Department of Computer Science, Davis, CA 95616, September 1995.
- [49] *Real Secure 2.5 User Manual*, Chapter 6. Internet Security Systems, Atlanta, GA. [available at <http://download.iss.net/manuals/rs25.tar.Z>]
- [50] Rootshell Website. <http://www.rootshell.com>. 1999.
- [51] Rootshell Website. <http://www.rootshell.com/archive-j457nxiq3gq59dv/199707/perl-ex.sh.html>. August 26, 1996.
- [52] Rootshell Website. http://www.rootshell.com/archive-j457nxiq3gq59dv/199707/solaris_ps.txt.html. June 11, 1997.
- [53] Rootshell Website. <http://www.rootshell.com/archive-j457nxiq3gq59dv/199707/fdformat-ex.c.html>. March 24, 1997.
- [54] Rootshell Website. http://www.rootshell.com/archive-j457nxiq3gq59dv/199711/sol_syslog.txt.html. November 6, 1997.

- [55] Rootshell Website. <http://www.rootshell.com/archive-j457nxiqi3gq59dv/199801/beck.tar.gz.html>. Jan 1, 1998.
- [56] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199805/xterm_exp.c.html. June 4, 1998.
- [57] Rootshell Website. <http://www.rootshell.com/archive-j457nxiqi3gq59dv/199806/mscan.tgz.html>. June 24, 1998.
- [58] Saint Website. <http://www.wvdsi.com/saint>. 1998.
- [59] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, K. Levitt, C. Wee, R. Yip, D. Zerkle, and J. Hoagland. GridS—a graph based intrusion detection system for large networks. 1996. [available at <http://seclab.cs.ucdavis.edu/arpa/grids/welcome.html>]
- [60] Sun Microsystems Security Bulletin: #00138. <http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/138>. 17 April, 1997.
- [61] Sun Microsystems Security Bulletin: #00140. <http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/140>. 14 May, 1997.
- [62] Sun Microsystems, Solaris Security Website. <http://www.sun.com/solaris/2.6/ds-security.html>. May 1999.
- [63] Sundaram, A. “In Introduction to Intrusion Detection”, Crossroads: The ACM Student Magazine, 2(4). 1996.
- [64] Daniel Weber. A Taxonomy of Computer Intrusions. Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.
- [65] Seth Webster. The Development and Analysis of Intrusion Detection Algorithms. Master’s Thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

(D) Art submitted by Applicant - Entered

The following items (1) – (7) listed below are hereby entered as evidence submitted to the Examiner as noted on an IDS transmission received by the USPTO on September 09, 2002.

- (1) Copy of EP Patent Number 1081918 ("Hinde et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (2) Copy of EP Patent Number 0991244 ("Chapman et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (3) Copy of WIPO Patent Number 200131852 ("Burnett et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (4) Copy of WIPO Patent Number 200067443 ("Nilsen"). This evidence was entered into the record by the Applicant on 09-09-2002.
- (5) Copy of WIPO Patent Number 200028431 ("Jacobs et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (6) Copy of International Search Report of International Patent Number PCT/US 01/47067. This evidence was entered into the record by the Applicant on 09-09-2002.
- (7) Copy of NPL- Self-Help for Bugs in the Field, Circuit Cellular, The Magazine for Computer Applications XP-002208351. This evidence was entered into the record by the Applicant on 09-09-2002.

Copies of all References follows.

//



(11)

EP 1 081 918 A2

(12)

EUROPEAN PATENT APPLICATION

(43) Date of publication:
07.03.2001 Bulletin 2001/10

(51) Int Cl.⁷ H04L 29/06

(21) Application number: 00307357.4

(22) Date of filing: 25.08.2000

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

- Wilcock, Lawrence
Malmesbury, Wiltshire SN16 9TV (GB)
- Low, Colin
Wotton-U-Edge, Gloucestershire GL12 7LT (GB)

(30) Priority: 04.09.1999 GB 9920834

(71) Applicant: Hewlett-Packard Company
Palo Alto, California 94304-1112 (US)

(74) Representative
Lawrence, Richard Anthony et al
Hewlett-Packard Limited,
IP Section,
Building 3,
Filton Road
Stoke Gifford, Bristol BS34 8QZ (GB)

(72) Inventors:
• Hinde, Stephen John
Redland Bristol BS6 7DH (GB)

(54) Providing secure access through network firewalls

(57) To enable controlled, secure connections using a versatile protocol such as TCP/IP to be established through a firewall and proxy server, the versatile protocol is tunnelled using HTTP. Client to server communications are effected using an HTTP POST operation. Server to client communications are effected using an HTTP GET operation to establish a tunnelled socket;

this socket is closed within an interval less than any timeout imposed by the proxy server and immediately re-established by another GET operation, irrespective of whether data continue to be pending for communication to the client. A globally-unique ID is included in each POST and GET message to enable related messages to be recognized.

Client ← Service

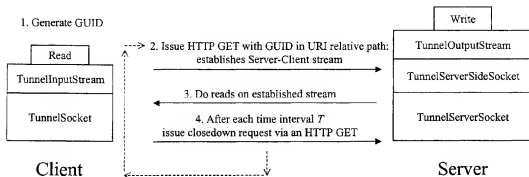


Fig.5

EP 1 081 918 A2

DescriptionTechnical Field

- 5 **[0001]** This invention relates to methods and apparatus for providing secure access between a service external to a network firewall (such as a web page server) and a client internal to the firewall (such as a web browser).

Background Art

- 10 **[0002]** Facilities for global access to information, such as the World Wide Web, are undergoing immense growth and expansion, both in volume of data transferred and in sophistication of tools, such as web browsers, for accessing those data. This in turn requires the flexibility and capability of the data network infrastructure to be increased, without at the same time jeopardising its security features. Often these objectives conflict.

- 15 **[0003]** Web based applications are "client-server" in nature. A client-based browser runs on an end user's computer, communicating via network links with a web server operated by a service provider. The server provides data defining the content of web pages which are rendered and displayed by the user's browser; typically the web page content is defined using a markup language such as Hypertext Markup Language (HTML). The communications between the browser and the web server are conducted in accordance with a protocol called Hypertext Transfer Protocol (HTTP), defined in the Internet Engineering Task Force's RFC 1945. HTTP is a simple text-based protocol that has the peculiar
20 quality that messages conforming to it are trusted and allowed to pass around the intranets, intranets and extranets that make up today's "Internet". This Internet is really a series of closely coupled networks linked together through "firewalls": network nodes that allow controlled, restricted access between two networks. However the ability to "browse the world wide web" is seen as a universal common denominator, and as such HTTP messages are allowed to pass through these firewalls unchecked. There have been a number of enhancements to the HTTP protocol: the description
25 herein relates by way of example to the basic version, HTTP/1.0, which is supported universally and with which later versions of HTTP are backwards compatible. Nonetheless the invention is not limited to use with HTTP/1.0 or indeed any other specific version of HTTP, and the claims hereof should be construed accordingly.

- [0004]** Web pages defined using the markup language can be enhanced by the use of code written in the Java programming language; this allows dynamic content and interactivity to be added to web pages. Java components can
30 run either on the server end of the network connection, as "servlets", or on the client browser machine, in which case they are known as "applets". Many potential security problems were envisaged with the introduction of applets, such as "viruses" and "Trojan horses", so a tight set of security restrictions were imposed on what applets could and could not do. To this end applets are executed on the client machine in a controlled environment called a "sandbox". This sandbox defines how the applet can interact with the resources available in the computational platform the applet is
35 running on, via a limited application programming interface (API). For example, the applet typically cannot interact with the local disk, nor connect to other computers on the network in unrestricted fashion. However the applet can typically connect back to the web server it was served from, although by way of an HTTP connection only. References to Java herein relate by way of example to Java/1.1 which is widely deployed; later versions of Java provide enhanced network support, but are backwards compatible with this base version.

- 40 **[0005]** The first generation of web content was mainly very static in nature - like pages from a magazine with text and pictures. The second generation of web content became increasingly dynamic, providing a user interface for applications, such as database queries. The third generation of web content is becoming increasingly interactive, with real-time communications, such as video, text chat and Internet Telephony, being added as an integral part. Internet-based client-server applications normally operate by opening "sockets" between the client and server, using Transac-
45 tion Control Protocol/Internet Protocol (TCP/IP). TCP/IP sockets provide bi-directional, reliable communications paths. These can be used to implement dynamic or interactive client-server based applications, such as are required by the second and third generations of web content.

- [0006]** However, these more sophisticated applications pose a problem, in that the communications protocols they often require for interaction with the web server, such as TCP/IP, are intentionally barred by firewalls and proxy servers.
50 It is therefore an object of this invention to provide clients, such as Java applets in a sandbox, with some controlled ability (within the context of a specific web-based application) to interact through a firewall with other resources, such as web servers, using protocols in addition to HTTP.

Disclosure of Invention

- 55 **[0007]** According to one aspect of this invention there is provided a method of permitting secure access between a service external to a network firewall and a client internal to the firewall, comprising the steps of:

- (a) effecting an HTTP GET operation or equivalent thereof (which may be in a protocol functionally equivalent to HTTP) from the client to establish a communications socket for communicating data from the service to the client;
 - (b) after a predetermined interval effecting another GET operation or equivalent thereof to close the communications socket, irrespective of whether access between the service and the client is required to continue; and
- 5 - repeating steps (a) and (b) while access between the service and the client is required to continue.

[0008] The HTTP Tunneling Socket described herein as an example of this invention is a socket that operates on top of the HTTP protocol. For interaction with a Java-based application it provides the standard APIs as defined for a socket in the Java/1.1 API specification. For interaction with the network it uses the HTTP/1.0 protocol. This technique

10 in which one protocol is carried in messages conveyed over another protocol is known as tunneling. The use of tunneling has two benefits. It allows the tunnel to be executed in the browser sandbox, since the APIs for accessing an HTTP connection to a web server do not have any security restrictions. Secondly it allows the client-server connection to traverse firewalls between intranets and internets, as HTTP is the protocol that all firewalls allow to pass through them.

[0009] The Java applet and the server are thus provided with a connection protocol (such as TCP/IP) which is more versatile and capable than the HTTP connection to which they would normally be limited by the firewall and/or proxy server. However, the restrictions imposed by the sandbox prevent malicious or inappropriate access to the client machine's resources through the tunnelled protocol.

15

Brief Description of Drawings

20

[0010] A method and apparatus in accordance with this invention, for permitting secure access between a client behind a firewall and a service provided by a node (such as a server) external to the firewall, will now be described, by way of example, with reference to the accompanying drawings, in which:

- 25 Figure 1 is a schematic illustration of circumstances in which the invention can be used;
 Figure 2 shows an outline of an implementation of the invention in the Java programming language;
 Figure 3 illustrates the basic principles of operation of the invention;
 Figure 4 shows in more detail the mechanism for communications from the client to the server; and
 Figure 5 shows in more detail the mechanism for communications from the server to the client.
- 30

Best Mode for Carrying Out the Invention, & Industrial Applicability

[0011] There are in principle two aspects involving firewalls to be considered when establishing a connection between an applet in a user's terminal and a web server. The first aspect is the case of a firewall-protected web server to which

35 a browser running on a machine outside the firewall is trying to connect. This aspect does not present a serious problem since the web-server owner can configure the firewall to allow this connection to be made.

[0012] The second aspect is the case of a web browser machine in a corporate intranet trying to connect through the corporate firewall to a web server on the public Internet, as illustrated in Figure 1. The problem here is that the web server owner has, by design, no control over the firewall. The complete scenario is that the browser 10 running java applet code on the user's terminal will connect initially to a machine 12 called a proxy within the corporate intranet 14. This proxy makes the required connections through the firewall 16 to the internet-based web server 18 on behalf of the browser 10, without exposing sensitive information about the browser and its host terminal to view in the public internet. The proxy 12 also performs various security checks, for example to ensure that the requests are valid HTTP requests.

40

[0013] However, because of the restriction to the use of HTTP, the web server is unable to provide desirable services to the client, such as real-time audio, video or interactive chat, which are possible with the use of more powerful protocols such as TCP/IP. These more powerful protocols are normally barred from unrestricted use because of the threat they pose of enabling inappropriate or intrusive access to resources within the corporate intranet.

45

[0014] This problem is overcome by using the invention to establish a "tunnel" through the firewall, carried by the HTTP messages, and thus open communications sockets which the Java applet and the web server can use to communicate with one another. In this way the web server is enabled to use resources on the browser's host machine, albeit still within the tight constraints imposed by the Java sandbox.

50

[0015] The tunnel consists of two Java programming classes, one (called the TunnelSocket) which implements the "Socket" class as defined in the Java/1.1 specification, and another (called TunnelServerSocket) which implements the "ServerSocket" class defined in that specification. As shown in Figure 2, client-based Java code can access the standard APIs (such as open, read, write, close, etc.) provided by a class conforming to the Socket class definition to communicate with another Java application, such as one running on the web server. The server-based Java application can likewise use the same Socket APIs, plus the standard APIs such as accept etc. provided for servlets by the

55

ServerSocket class, to communicate with the client-based Java applet. Messages between the Socket class and the ServerSocket class are carried within HTTP messages which traverse the firewall between the client and the server.

[0016] The HTTP tunnel uses the underlying simple text-based HTTP protocol to produce a reliable connection-based protocol between the TunnelSocket and TunnelServerSocket classes. Basic HTTP transactions use one of two methods in the HTTP request message. The GET method is designed to "get" a web page from a web server and the POST method is used to send data to a remote web resource or web application. In implementing the present invention, a GET based protocol is used for the client to server direction, and a POST based protocol is used for the reverse, server to client, direction, as shown in Figure 3. In the event that HTTP is supplanted by a functionally equivalent protocol, it is envisaged that the invention could be used with operations in that protocol equivalent to the HTTP GET and POST operations.

[0017] A single connection between the TunnelSocket and TunnelServerSocket classes is likely to comprise multiple HTTP transactions. To associate these transactions together, a numeric Globally Unique ID (GUID) is generated using a combination of random numbers, network address and time of day so that each GUID generated is a unique quantity both in relation to time and location in the network. This GUID is incorporated in each HTTP request, both GET and POST, so that the HTTP protocol can recognize that they relate to the same communications socket. Specifically, the GUID is passed as the HTTP Uniform Resource Indicator (URI) (the field used to specify a document name in normal use of HTTP).

[0018] Client to server connection is implemented using the POST HTTP operation. Each time a write is performed to the client socket (TunnelSocket class) the request is packaged up and sent as the payload of an HTTP POST message, including the relevant GUID, as shown in Figure 4. Each such write operation sends a separate POST message through the firewall.

[0019] The server to client direction is more complex because of restrictions imposed by the proxy server 12 and the firewall 16. As shown in Figure 5, the client issues an HTTP GET message which tells the server that a new tunneled socket is being established. The HTTP GET request will generate a temporary server-client socket that allows data to be written from the server to the client. However as a security precaution most proxy server implementations will close down such a server-client socket after a time interval P , typically of the order of minutes in duration. Accordingly the tunnel established in this invention periodically itself forces a pre-emptive close down of the connection at some periodicity T which is shorter in duration than the interval P . This is accomplished by issuing another GET request, to force the TunnelServerSocket class to close the existing socket to the client and immediately establish a new socket. It should be noted that this closure (and subsequent reopening) are performed even though data are still pending to be transferred from the server to the client. This approach avoids the overhead of previous techniques, in which the client polls the server at some preselected polling interval usually of the order of 20 milliseconds. These frequent polls are equivalent to multiple web page requests, so the server receives a large number of web hits per second, imposing a significant overhead and severely restricting the scalability of such prior techniques to handle large numbers of clients simultaneously.

[0020] An illustration of the implementation of the invention is provided below, in the form of pseudo-code, which emphasizes the significant aspects of the implementation but for the sake of clarity omits minor details which, although required in practice, are well within the capability of a person skilled in the art. Likewise the existence is assumed of certain subsidiary functions, such as guidFactory for creating a GUID and createFIFOOutputStream for reserving and configuring memory for use as a first-in-first-out (FIFO) buffer; again these functions are individually well known in the art, so their internal details do not need to be specified. Depending on the particular manner of implementation additional parameters may be included, for example, in some function calls; this is indicated by an ellipsis (...).

1. Server to Client direction.1a. TunnelSocket InputStream Code – runs on client

```

5
//
// Open an Input Stream to the Server
// This is the Server to Client direction of the Tunnel Socket.
10
//
// serverAddress - Internet address e.g. fred.hpl.hp.com:
// standard socket
// Port number is a port number on the server machine to
// connect to: standard socket
15 public open(serverAddress, portNumber) {

    // Generate a new Globally unique id for the new socket.
    // The id is unique across the whole network and across
    // time.
20    // It is generated using a random number generator, the
    // node's network address and the current time.
    guid = guidFactory();

    // Create a Fifo stream that is used by the application code
    // to read data.
    // The underlying protocol reads data off the wire into this
    // Fifo ready for the application.
    fifoInputStream = createFifoInputStream();

30    // Issue an HTTP/1.0 GET request to the server
    // The on-the-wire format is:
    // GET <guid>?command=establish
    // Content-Type=application/octet-stream
    inputStream = issueHttpRequest(uri=guid,
35    queryString="command=establish",
    Content-Type="application/octet-stream",
    address=serverAddress, port=port);

40    // Start a timer thread that is invoked at a periodicity of
    // GETTimeOut seconds.
    // Typically GETTimeOut is a few minutes in duration - but
    // it must be less than the time out period of the proxy.
    // The function switchGETStream() is called at the
    // periodicity with its own thread of execution.
45    startPeriodicTimeOutDemon(switchGETStream(), GETTimeOut);

}

50

// Send a new request to the server telling it to close down
// the current stream and establish a new stream
55 // periodically

```

```

private switchGETStream() {

    // Lock the Input Stream to prevent any race conditions with
    //   readInputStream
5    lock(inputStreamLock)

    newInputStream = issueHttpRequest(uri=guid,
        queryString="command=switch",
10    ContentType="application/octet-stream",
        address=serverAddress, port=port);

    // Drain the contents of the stream until it is closed by
    //   the remote end
15    while (read(InputStream, FifoStream) != EOF);
        writeToFifo(FifoInputStream);

    inputStream = newInputStream();
    unlock(inputStreamLock);

20    return();

}

25

// Read data from the stream via the Fifo - the Fifo buffers
//   data up ready to be read
public readInputStream(..., buf, len) {
30    ...
    // if there isn't enough data in the Fifo to satisfy this
    //   read request - top it up from the wire.
    if (!FifoStream.length() > len) {

35        //Transfer data from the stream established to the Server
        lock(inputStreamLock);
        writeToFifo(FifoInputStream, read(InputStream,
            len - FifoStream.length());
40        unlock(inputStreamLock);
    }

    // return data in from Fifo
    return(read(FifoInputStream, ..., buf, len));

45    }

50    public close() {

        // Stop timer thread
        stopPeriodicTimeOutDemon(switchGETStream());

55

```

```
// Mark stream as closed  
markInputStreamClosed();
```

5

```
}
```

10

15

20

25

30

35

40

45

50

55

1b. TunnelServerSocket OutputStream – runs on the server

```

5      // Server side Protocol engine handling all requests the server receives

      // Sit waiting on a Serversocket for socket connections to come
      // in ... just standard socket call
      while ( (socket = accept()) still open) {

10         httpheader = readHttpHeader(socket);

         if ( httpHeader.type == "GET" &&
             httpHeader.command == establish) {

15             // New socket request
             // Open an output Stream
             outputStream = socket.getOutputStream();

             // Add an entry to the database of opened Tunnel Sockets
20             addEntryTunnelSocketHashDataBase(socket, outputStream,
                 httpHeader.guid);

             } else ( httpHeader.type == "GET" &&
                 httpHeader.command == "switch") {

30                 // Lookup tunnelSocket
                 tunnelSocket =
                     lookupEntryTunnelSocketDatabase(httpHeader.guid);

                 // Request to switch stream generated by the TunnelSocket
                 // timer on the client
                 // Close the current output stream
                 // Take lock to avoid conflict with
35                 // tunnelOutputStreamWrite
                 lock(tunnelSocket.outputStreamLock);
                 close(tunnelSocket.outputStream);

                 // establish a new output stream on this new connection.
40                 tunnelSocket.outputStream = socket.getOutputStream();
                 unlock(tunnelSocket.outputStreamLock)
             } else (httpheader.type == "POST" )
                 // See pseudo-code fragment included below with Server-
                 // Client direction.
                 doPOST();

50         }

         tunnelOutputStreamWrite(buf, len) {
             ...
             // Take lock to avoid conflict with "switch" in
             // tunnelServerAccept()
             lock(outputStreamLock)
55         }

```

```
write(outputStream, buf, len);  
unlock(outputStreamLock)
```

```
}
```

5

.....

10

15

20

25

30

35

40

45

50

55

2. Client-Server direction.**2a. Client side TunnelSocket OutputStream**

```

5      public write(buf, length, ...) {
10          // Issue an HTTP/1.0 POST request to the server
          // The on the wire format is:
          //   POST <guid>
          //   Content-Type=application/octet-stream
          //   Content-Length=<length>
15          inputStream = issueHttpRequest(uri=guid,
              Content-Length=length,
              Content-Type="application/octet-stream",
              address=serverAddress, port=port, payload=buf);
20      }

```

2b. Server side TunnelServerSocket InputStream

```

25      public readServerSideInputStream(buf, len, ...) {
30          return(readFromFifo(FifoServerSideInputStream, buf, len,...);
          }
35
36      ...
40      // On receiving a POST request (see 1b above)
      // copy the payload into the Fifo buffer ready for the
      // readServerSideInputStream to read it.
      doPOST() {
45          writeToFifo(FifoServerSideInputStream,
              httpRequest.readPayload(), httpRequest.contentLength);
          }
      ....
50

```

Claims

- 55 1. A method of permitting secure access between a service external to a network firewall and a client internal to the firewall, comprising the steps of:

- (a) effecting an HTTP GET operation or equivalent thereof from the client to establish a communications socket for communicating data from the service to the client;
 - (b) after a predetermined interval effecting another GET operation or equivalent thereof to close the communications socket, irrespective of whether access between the service and the client is required to continue; and
 - repeating steps (a) and (b) while access between the service and the client is required to continue.
2. The method of claim 1, wherein the predetermined interval is less than another interval after which client software enforces termination of a communications socket established by a GET operation or equivalent thereof.
3. The method of claim 1 or claim 2, wherein information is transferred from the client to the service by an HTTP POST operation or equivalent thereof.
4. The method of any one of the preceding claims, wherein successive messages transferred between the client and the service are identified by a globally-unique identification created by the client and communicated to the service.
5. The method of claim 4, wherein the globally-unique identification is communicated via an HTTP GET or POST operation or equivalent thereof.
6. The method of claim 5, wherein the globally-unique identification is communicated in a URI relative path component.
7. The method of any one of the preceding claims, wherein communications with the client traverse a proxy service located on the client side of the firewall.

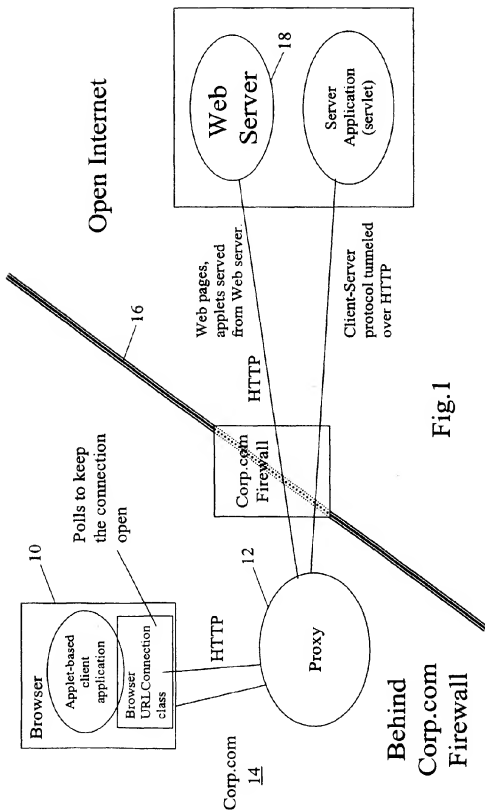
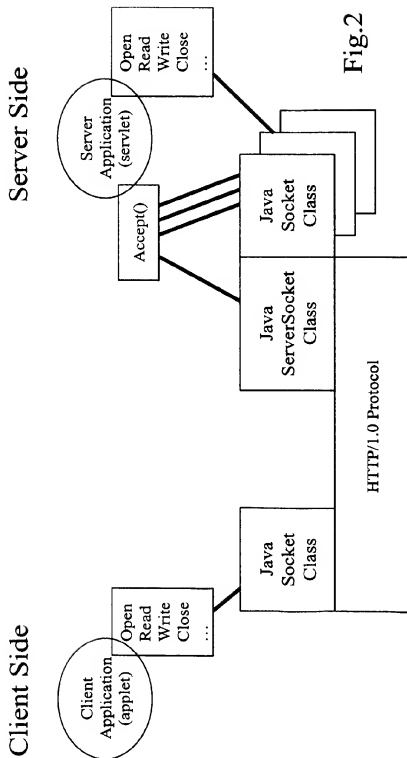


Fig.1



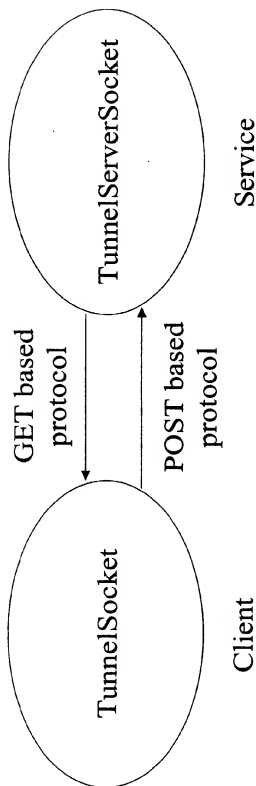


Fig.3

Client → Service

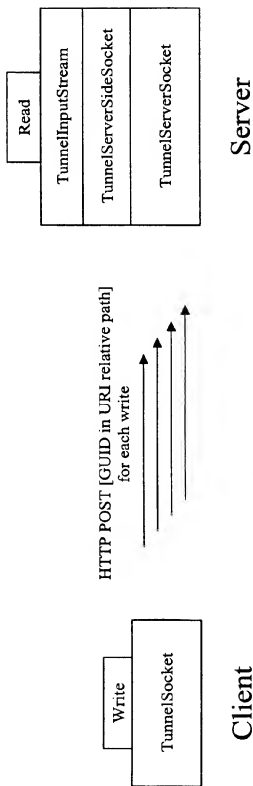


Fig.4

Client ← Service

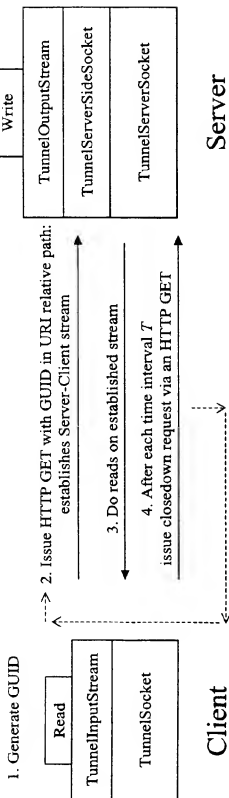


Fig.5

(12) **EUROPEAN PATENT APPLICATION**

(88) Date of publication A3:
17.12.2003 Bulletin 2003/51

(51) Int Cl⁷: **H04L 29/06**

(43) Date of publication A2:
07.03.2001 Bulletin 2001/10

(21) Application number: **00307357.4**

(22) Date of filing: **25.08.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
 Designated Extension States:
AL LT LV MK RO SI

• Wilcock, Lawrence
 Malmesbury, Wiltshire SN16 9TV (GB)
 • Low, Colin
 Wotton-U-Edge, Gloucestershire GL12 7LT (GB)

(30) Priority: **04.09.1999 GB 9920834**

(71) Applicant: **Hewlett-Packard Company,**
A Delaware Corporation
Palo Alto, CA 94304 (US)

(74) Representative: **Lawrence, Richard Anthony et al**
Hewlett-Packard Limited,
IP Section,
Building 3,
Filton Road
Stoke Gifford, Bristol BS34 8QZ (GB)

(72) Inventors:
 • Hinde, Stephen John
 Redland Bristol BS6 7DH (GB)

(54) **Providing secure access through network firewalls**

(57) To enable controlled, secure connections using a versatile protocol such as TCP/IP to be established through a firewall and proxy server, the versatile protocol is tunnelled using HTTP. Client to server communications are effected using an HTTP POST operation. Server to client communications are effected using an HTTP GET operation to establish a tunnelled socket;

this socket is closed within an interval less than any timeout imposed by the proxy server and immediately re-established by another GET operation, irrespective of whether data continue to be pending for communication to the client. A globally-unique ID is included in each POST and GET message to enable related messages to be recognized.

EP 1 081 918 A3

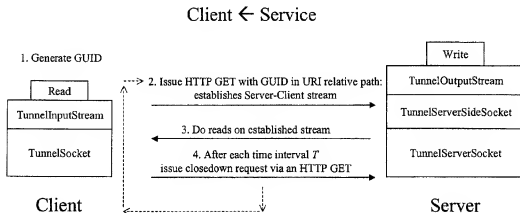


Fig.5



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 00 30 7357

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
P,X	US 6 169 992 B1 (APPELBAUM MATTHEW A ET AL) 2 January 2001 (2001-01-02) * abstract * * column 3, line 21 - column 5, line 62 * * column 18, line 30 - column 37, line 35 *	1-7	H04L29/06
A	--- HUNT R: "Internet/Intranet firewall security-policy, architecture and transaction services" COMPUTER COMMUNICATIONS, BUTTERWORTHS & CO. PUBLISHERS LTD, GB, vol. 21, no. 13, 1 September 1998 (1998-09-01), pages 1107-1123, XP004146571 ISSN: 0140-3664 * abstract * * page 1107, right-hand column, line 18 - page 1108, left-hand column, line 47 * * page 1116, left-hand column, line 30 - page 1117, left-hand column, line 5 * * page 1117, right-hand column, line 31 - page 1119, left-hand column, line 4 * -----	1-7	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 29 October 2003	Examiner Adkhis, F
CATEGORY OF CITED DOCUMENTS		T: theory or principle underlying the invention E: earlier patent document, but published on, or after the filing date D: document cited in the application L: document cited for other reasons A: technological background O: non-written disclosure P: intermediate document	
X: particularly relevant if taken alone Y: particularly relevant if combined with another document of the same category A: technological background O: non-written disclosure P: intermediate document		8: member of the same patent family, corresponding document	

EPO FORM 1200 (2.02.99) (C1)

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
05.04.2000 Bulletin 2000/14

(51) Int Cl.⁷ **H04L 29/08, H04L 29/06,
H04L 12/56**

(21) Application number: **99307651.2**

(22) Date of filing: **28.09.1999**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE**
Designated Extension States:
AL LT LV MK RO SI

(72) Inventors
• **Chapman, Alan Stanley John**
Kanata, Ontario K2K 1V5 (CA)
• **Kung, Hsiang-Tsung**
Lexington, MA 02173 (US)

(30) Priority: **30.09.1998 CA 2249152**

(74) Representative: **Funnell, Samantha Jane et al**
Hepworth Lawrence Bryer & Bizley
Merlin House
Falconry Court
Bakers Lane
Epping, Essex CM16 5DQ (GB)

(71) Applicant: **Nortel Networks Corporation**
Montreal, Quebec H2Y 3Y4 (CA)

(54) **Apparatus for and method of managing bandwidth for a packet based connection**

(57) Flow control of packet based traffic by window is known. Novel modification is described which causes the flow control mechanism to reduce sending rate to some configured number rather than just reducing it by a fixed amount such as one half. The description also shows how the flow control mechanism can be constrained to a maximum rate. The configured numbers will assure that the connection can always run at a minimum rate but not more than a maximum rate. If the

guaranteed minimum bandwidth is known and the round trip time between the end points is known or has been calculated, then the sender node needs only reduce its window to that which corresponds to a sending rate equal to that configured number. In this way the protocol will still probe for extra, opportunistic bandwidth but will be able to maintain the minimum rate. In a similar way a window that corresponds to the maximum rate can be calculated and used to constrain the maximum rate of sending.

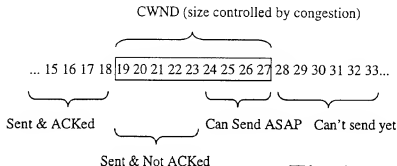


Fig 1

Description**Field of Invention**

[0001] The invention resides generally in the field of digital data transmission through a network. In particular, it relates to an apparatus for and method of transmitting digital data in streams of packets while observing guaranteed minimum and/or maximum bandwidth allocations.

Background of Invention

[0002] In contrast to circuit based transport systems, a packet based transport system allows the access bandwidth to be dynamically allocated. Remote nodes can be represented as logical ports but there is no commitment of bandwidth when this is not needed. The physical access link is fully available for traffic to any destination. In a packet transport system, virtual pipes are provided between any two transport access points. These pipes may be guaranteed some minimum rate of transmission but certainly it is required that an access point can make opportunistic use of spare bandwidth up to some maximum amount. Frame relay as a packet based transport allows more efficient use of bandwidth by permitting statistical multiplexing of data streams, thus allowing it to exploit unused bandwidth. However, there is no mechanism (protocol) to ensure reliable delivery of the frames and under congestion conditions frames are discarded and the higher layer protocols must compensate for the loss. The discard of frames is not sensitive to the impact on the higher layer protocols and the frame flows do not directly adapt to the network conditions. ATM with an effective flow control can provide a lossless but dynamic transport. However, it relies on some reasonable level of complexity at the transport switching points to achieve the flow control, the effectiveness of which has not yet been proven in the field. ATM without flow control requires that cells be discarded under congestion conditions and this discard should be aware of packet boundaries and the impact on the higher layer protocol such as TCP protocol. These issues are just beginning to be understood.

[0003] It is commonly understood in the field of the present invention that a layer under the networking layer is called "transport" layer and provides pipes between networking layer nodes. This is in contrast to the layered model of the OSI (open systems interconnect) in which the transport layer resides upon the network layer, which in turn sits on top of the data link layer. The data link layer provides similar functionalities to those of the transport layer of the present description. Throughout the specification, the former designation is used.

[0004] Therefore in the TCP/IP model, IP layer resides under TCP layer. The IP layer is the network layer in which IP (Internet protocol) runs. An internetwork differs from a single network, because different parts may

have wildly different topologies, bandwidth, delays, packet sizes, and other parameters. The TCP layer is the transfer layer in which the TCP (transmission control protocol) runs. The TCP has been used for ensuring reliable transfer of byte stream data between two end points over an internetwork, which may be less reliable. TCP allows a shared and adaptive use of available bandwidth of a transmission link between two end points. It does this by having the sender gradually increase the rate of sending until a packet is lost whereupon it reduces its rate significantly and repeats the gradual increase. Thus, TCP tends to give connections their proportional share of the available bandwidth on a link although different connection characteristics can cause large variations in the sharing.

[0005] In Internet terminology, aggregating traffic streams by encapsulating them into a single IP stream is often called tunneling. Applicant's compending patent application Serial No. 09/066,888 filed on Apr. 28, 1998 describes an invention, which re-uses TCP in a packet based transport to provide TCP tunneling which can conveniently be called "TCP trunking". The use of TCP provides for reliable delivery of data between two transport access points while permitting that transport to offer elasticity and bandwidth sharing. Aggregating traffic streams into TCP tunnels reduces the size of buffers and tables in the transport switches. TCP is well suited to the use of first-in-first-out queues and allows simple implementations at the switching nodes. TCP is also inherently provides for resequencing of out-of-order packets which can occur when switching nodes spread load over multiple links.

[0006] It is expected that in future networks, particularly those using TCP trunking between aggregation points, it will be required to assure a minimum sending rate for the connection while still allowing the connection to probe for more throughput if it is available. This opportunistic acquisition of bandwidth might be limited to some maximum. TCP currently has no capability to support a guaranteed minimum bandwidth or a maximum permitted bandwidth but this will be essential as IP networks introduce virtual private networks with performance guarantees. The present invention offers a good solution to such a problem.

[0007] It should be noted that in a carrier owned transport network the TCP functionality would be in transport access points rather than host computers, the variability of paths will be low and parameters such as connection round trip time will be very stable. These features make it much easier to envisage modifications to the TCP protocol for these networks. However, the use of this invention can be more generally applied to other TCP hosts and other sliding window protocols.

[0008] Many flows on today's networks carry only a small amount of information, say ten or twenty packets, and the rate adaptation feature of TCP is not as relevant as the reliability feature. Providing reliability without incurring TCP time-outs on the loss of a single packet

would greatly improves user throughput and network efficiency.

Objects of Invention

[0009] It is therefore an object of the invention to provide sliding window based flow control for point-to-point connections in a packet network, which constrain the minimum and/or maximum bandwidths available to that connection.

[0010] It is another object of the invention to provide modifications to the TCP protocol to provide for constraining the minimum and/or maximum bandwidths achieved by the connection.

[0011] It is yet an object of the invention to ensure the absence of TCP retransmission timeouts for flow with small windows by maintaining a minimum rate of sending.

Summary of Invention

[0012] Briefly stated, in accordance with one aspect, the invention is directed to a method of sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed minimum bandwidth and a maximum permitted bandwidth. The method comprises steps of (1) calculating a congestion window hereinafter called C-WND of the connection, (2) calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth window hereinafter called MAX-WND. The method further includes steps of (3) determining if the MIN-WND or MAX-WND is invoked on the connection, based on their relationship with C-WND and (4) allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.

[0013] According to another aspect, the invention is directed to an apparatus for sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed minimum bandwidth and a maximum permitted bandwidth. The apparatus comprises a flow control module for controlling a flow of packets into the connection in response to acknowledged packets, a congestion window arithmetic module for calculating a congestion window hereinafter called C-WND of the connection and a bandwidth monitoring window arithmetic module for calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth window hereinafter called MAX-WND. The apparatus further includes control logic module for determining if the MIN-WND or MAX-WND is invoked on the connection, a transmitter for transmitting a series of

packets of data into the connection and a controller for allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.

Brief Description of Drawings

[0014]

Figure 1 is an illustration showing the nature of the sliding window algorithm.

Figure 2 shows mechanisms of acknowledgements.

Figure 3 shows that the window is inflated and moved when a non-duplicate ACK is received.

Figure 4 shows the window when a packet is lost.

Figure 5 shows the nature of MIN-WND according to an embodiment of the invention.

Figure 6 shows the MIN-WND when a duplicate ACK is received.

Figure 7 shows the MIN-WND when a non-duplicate ACK is received.

Figure 8 shows the nature of MAX-WND according to an embodiment of the invention.

Figure 9 shows the MAX-WND when a duplicate ACK is received.

Figure 10 shows the MAX-WND when a non-duplicate ACK is received.

Figures 11-14 show the relationships of various windows.

Figure 15 is a block diagram of a TCP node.

Detailed Description of Preferred Embodiments of Invention

[0015] When providing services with bandwidth guarantees, a packet transport network should be able to emulate the circuit-based mesh in that a defined minimum bandwidth can be allocated between any pair of nodes. However, unused bandwidth should be made available to other flows in a dynamically shared fashion so that a flow can opportunistically exceed its minimum. In some cases it is also useful to implement a maximum limit on how much extra bandwidth a pair of nodes can use.

[0016] The conventional IP network implements bandwidth sharing among host machines using the transport control protocol (TCP). In TCP the sender (sender host machine) constantly tests the network to see if more bandwidth is available and uses the loss of a packet determined by sequence numbers of TCP packets as an indication to decrease its rate. Any lost packets are sent again so that there is a reliable flow of traffic. The loss of too many packets can cause the TCP connection to enter the timed out state. Consecutive timeouts are increased in an exponential way until eventually the connection is closed.

[0017] The general characteristic of TCP is that it is

self-clocking. That is to say, the sender will wait for an acknowledgment from the receiver for the packets already sent before sending more packets. If the sender waited for each individual packet to be acknowledged then the maximum rate that the connection could achieve would be one packet per round trip time of the connection. To increase the sending rate while keeping the self clocking nature of the protocol, the sender is allowed to send some number of packets while waiting for an earlier packet to be acknowledged. This number of packets is called the window. The receiver itself may constrain the size of the window in order to limit its buffer requirement.

[0018] Each packet contains a sequence number, which increases according to the number of bytes transmitted. The receiver acknowledges packets using this numbering scheme and always acknowledges the latest packet received in correct sequence. It may acknowledge each packet individually or wait in order to reduce overhead (this is called Delayed ACK). It should definitely send an acknowledgment at least every second packet. If a packet is received which is not in correct sequence the receiver will immediately send an acknowledgment but the sequence number it acknowledges will be that of the last packet which was received in the correct sequence. It should be noted that the sequence number in a packet corresponds to the last byte in the packet and the acknowledgment contains the next expected in-sequence byte number and thus acknowledges all bytes up to that number. In general terminology a packet is acknowledged when the receiver reports that the next expected byte number is later than any bytes contained in that packet.

[0019] The maximum rate of sending on a TCP connection is equal to the window size divided by the round trip time of the connection. TCP will constantly try to increase its rate by increasing the window size. When a packet is lost the window size is reduced and the gradual increase is begun again. The current size of the window is called the congestion window (C-WND) and can vary between one packet and the maximum that the receiver is prepared to accept (R-WND receiver window).

[0020] Figure 1 shows the nature of the sliding window. The window reflects the data sent but not yet acknowledged as well as the amount of data that can still be sent without waiting for an acknowledgement. As a packet is acknowledged the window advances so that the left-hand side is equal to the earliest unacknowledged byte number. The right hand side of the window is equal to the highest byte sequence number that can be sent before the transmitter must wait for further acknowledgements. It should be noted that the receiver will only acknowledge bytes received which are in a complete sequence. Later bytes that have been received will not be acknowledged until all previous bytes have been received.

[0021] Packet loss is detected in one of two ways. If the sender does not get an acknowledgment within a

certain time (TCP retransmission time-out) it will assume that a packet has been lost and will reduce its C-WND size to one packet as well as resending the lost packet. If the sender sees multiple acknowledgments (called duplicate ACK) of the same packet it can decide that packet has been lost even before the retransmission time-out occurs. Many TCP implementations include this fast retransmission and recovery capability. The window size is cut in half and the lost packet is retransmitted. Avoiding time-out gives a great boost to perceived performance but it is only effective when the window is large enough to allow enough duplicate acknowledgments to be generated (usually three). This is shown schematically in Figure 2 in which packet 19 is lost and the receiver acknowledges reception of packets up to 18. Multiples of acknowledged packets indicate that the receiver still expects packet 19. For windows smaller than about five packets, it is not possible to guarantee that fast retransmission will be invoked. Many flows on today's networks carry only a small amount of information, say ten or twenty packets and never develop large windows. Thus, the loss of a single packet can force the connection into timeout.

[0022] There are two operations on the C-WND and they are shown in Figures 3 and 4 respectively

(a) Inflate Window (Figure 3)

When a non-duplicate ACK is received, C-WND is inflated by extending the window's right edge and moves to the right so that the first byte in the window is the earliest unacknowledged byte. The inflation factor is a function of the TCP implementation.

(b) Deflate Window (Figure 4)

When packet loss occurs the window is reduced in size by retracting the window's right edge and the packet is retransmitted.

[0023] The transport system is required to provide some minimum level of bandwidth for the total traffic between any pair of access points. As mentioned earlier, usually TCP will reduce its sending rate very aggressively when a packet is lost. It is envisaged however that TCP can be modified to cause it to reduce sending rate to some configured number rather than just reducing it by a fixed amount such as one half. The configured number will assure that the connection can always run at a minimum rate. It is also envisaged in some instances that TCP can be constrained to a maximum rate, less than it would achieve normally so that a connection would not occupy all the available bandwidth.

[0024] Therefore, if the guaranteed minimum bandwidth is known and the round trip time (RTT) between the end points is known or has been estimated, then the TCP sender node needs only reduce its window to that which corresponds to a sending rate equal to that configured number. In this way the protocol will still probe for extra, opportunistic bandwidth but will be able to

maintain the minimum rate. Similarly sending will be inhibited when the TCP window reaches a size corresponding to the maximum bandwidth.

[0025] This invention introduces the concept of overlay windows. This concept makes it easy to understand the design intent and permits the modification to be added without having to make substantial changes to the main body of standard TCP operating code.

[0026] According to one embodiment, TCP is modified to cause it to constrain its sending rate to be between some configured minimum and/or maximum numbers rather than between one packet and the receiver window size (R-WND). This modification is only needed at the TCP transmitter. The configured minimum number will assure that the connection can always run at a minimum rate and the configured maximum number prevents all the available bandwidth of the connection from being taken by a node pair. The modification to the TCP transmitter will also improve TCP's resilience in the sense that the connection will not experience exponentially increasing time out under packet loss. This improved resilience against packet loss is achieved without loading the network more than the desired guaranteed minimum bandwidth for the TCP connection, or one packet per the round trip time of the connection. As well as providing for a guaranteed minimum rate during the lifetime of a connection, this modification can also be enabled selectively to prevent time-out at the times when the window size of the connection is too small to allow fast retransmission and recovery.

[0027] As mentioned earlier, in normal TCP the sender is allowed to send some number of packets while waiting for an acknowledgment of an earlier packet and this number is referred to as the window. Arithmetically, one can see that the maximum rate that a connection can achieve is equal to the window size divided by the round trip time of the connection (RTT) in seconds. To assure a guaranteed minimum bandwidth (GMB bytes per second) it is necessary that the connection can always send at least GMB times RTT bytes in any RTT period and that the connection is not stalled by lost packets but will keep sending unless the normal keepalive process closes the connection.

[0028] According to one embodiment, the TCP transmitter uses following variables:

GMB: The guaranteed minimum bandwidth in bytes per second. This is a new, configured parameter.

MPB: The maximum permitted bandwidth in bytes per second. This is a new, configured parameter.

RTT: The estimated round trip time of the connection in seconds.

R-WND: The maximum window size acceptable to the receiver (as advertised by the receiver in conjunction with acknowledgements).

C-WND: The size of the congestion window as computed by the existing TCP algorithm.

MIN-WND: This is a sliding window based on the

guaranteed minimum bandwidth. This is a new, computed variable

MAX-WND: A sliding window based on the permitted maximum bandwidth. This is a new, computed variable.

Pkt: Packet size is the packet payload size currently used by the connection, in bytes

RTT: TCP retransmission time-out value is the period, in seconds, after which, in the absence of acknowledgments, unacknowledged packets will be retransmitted. (Typically RTT is 1 sec or greater.)

OwT: Out-of-window send timer, in seconds, defines the time after which a packet can be sent even if the state of the TCP congestion window would normally not allow it. (A suggested value of OwT is 0.2 secs.). This is a new configured parameter.

RsT: Resend timer period, in seconds, has a value larger of OwT, RTT,

Pkt/GMB but always smaller than RTT. This is a new, computed variable.

[0029] According to an embodiment of the invention, the guaranteed minimum bandwidth for a TCP connection is achieved as follows.

[0030] While the connection is open the transmitter can send one packet into the network if it is allowed by the sliding window advertised by the receiver and if any one of the following conditions are met:

C1. The transmitted packet is allowed by the normal TCP congestion window and not disallowed by MAX-WND

C2. The transmitted packet is allowed by MIN-WND

C3. RsT expires.

[0031] Figure 5 shows the nature of MIN-WND, which is very similar to C-WIND except that the size does not change according to congestion but is tied to the value of GMB and RTT.

[0032] There are two operations on the MIN-WND and they are shown in Figures 6 and 7

(a) Inflate Window (Figure 6)

When a duplicate ACK is received, MIN-WND is inflated by 1 Pkt by extending the window's right edge

(b) Reset Window Size and Move Right (Figure 7) When a non duplicate ACK is received, MIN-WND resets to its original size (GMB*RTT), and moves to the right so that the first byte in the window is the earliest unacknowledged byte.

[0033] As seen in the figures, when a duplicate acknowledgement is received the window is inflated by one packet. The reception of an acknowledgement shows that the receiver has received a packet even if it was not in sequence. The inflation of the window ensures that the connection can continue sending new packets even when an acknowledgement is missing. A

new packet will be sent for each duplicate acknowledgement received. This prevents the connection stalling but does not increase the number of packets in the network. However, as soon as a non-duplicate acknowledgement is received the window is reset to the normal size.

[0034] Similarly Figure 8 shows the format of MAX-WND which is identical to MIN-WND except that the size is based on the maximum permitted bandwidth.

[0035] Like the windows described thus far, there are two operations on the MAX-WND and they are shown in Figures 9 and 10:

(a) Inflate Window (Figure 9)

When a duplicate ACK is received, MAX-WND is inflated by 1 Pkt by extending the window's right edge.

(b) Reset Window Size and Move Right (Figure 10)
When a non duplicate ACK is received, MAX-WND resets to its original size (MPB+RTT), and moves to the right so that the first byte in the window is the earliest unacknowledged byte.

[0036] Figures 9 and 10 therefore show that the window is also inflated when duplicate acknowledgements are received and reset when a non-duplicate acknowledgement is seen.

[0037] Figures 11 to 14 show unmodified TCP algorithm and the overlay methods of modifying the TCP algorithm, according to embodiments of the invention. In these embodiments, the chosen window is defined as being the maximum acceptable window after taking into account the requirements of the overlay rules. Therefore, Figure 11 shows unmodified TCP where the chosen window is the normal congestion window and can have a value between 1 and R-WND. Figure 12 on the other hand shows how, when the highest sequence number within C-WND falls below that of MIN-WND, the GMB overlay takes effect and the chosen window is equal to MIN-WND. Similarly in Figure 13, when the highest sequence number of C-WND becomes greater than that of MAX-WND, the MPB overlay takes effect and chosen window becomes equal to MAX-WND. Figure 14 shows how MIN-WND and MAX-WND are overlaid on the normal TCP algorithm to provide the complete bandwidth control.

[0038] When the rate of sending is between the configured limits the normal TCP algorithm controls the rate. If the rate tends to fall below the minimum then MIN-WND comes into play. If the rate reaches the maximum then MAX-WND takes effect. The normal TCP mechanism is still in control of reliability and of elasticity within the configured limits. An overriding timer RST ensures that even when no acknowledgements are being received, a minimum rate of packets are still sent to stimulate acknowledgements and eventually retransmission without being stalled by TCP timeout.

[0039] Figure 15 illustrates schematically in block diagram a TCP node according to one embodiment of the

invention. The node is connected to a network and includes a transmitter and receiver of an IP module 20. The customer data 22 is processed by a TCP module 24 which forms the data into TCP packets before the transmitter send them into the network. In receive direction, of course, the module extracts the customer data and transfers it to the customer's terminal for outputting. The clock 26 generates clock signals which times a variety of operations of the node. The arithmetic module 28 is shown in a separate box which performs computations described thus far under control of the control logic 30. Controller 32 supervises the over-all operation of TCP module.

15 Assertions:

[0040]

A1. The TCP connection is allowed to transmit at least GMB*RTT bytes per RTT except while recovering from missing acknowledgements.

A2. The TCP connection will not stop transmitting for a period longer than RST under any circumstances of packet loss, until a normal TCP timeout causes the connection to close.

A3. The loading of the network by the transmitter is at most the maximum of GMB and Pkt/RST, unless more is allowed by the TCP congestion window within the limits of MPB. (RST is always greater than or equal to RTT)

A4. In the presence of sufficient bandwidth, the TCP connection is able to sustain a maximum rate of MPB*RTT bytes per RTT except while recovering from missing acknowledgements.

[0041] It should be noted that it is a necessary requirement that an admission process will limit the number of TCP connections sharing a network link, so that the sum of GMBs for all the TCP connections including TCP and IP header overheads is no more than the link bandwidth

[0042] It should also be noted that even when a guaranteed minimum bandwidth is not wanted for the whole duration of a flow, the application of this modification would improve performance for short flows of less than say ten or twenty packets by preventing TCP time-out which normally occurs after a single packet loss. In this case the modification could be enabled while the flow was in a fragile state and turned off once a certain number of packets were successfully sent

Claims

1. A method of sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed mini-

imum bandwidth and a maximum permitted bandwidth, comprising steps of:

- (1) calculating a congestion window hereinafter called C-WND of the connection;
 - (2) calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth hereinafter called MAX-WND;
 - (3) determining if the MIN-WND or MAX-WND is invoked on the connection, based on their relationship with C-WND; and
 - (4) allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.
2. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 1 wherein step (3) is performed by comparing either C-WND and MIN-WND or C-WND and MAX-WND
 3. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 2 comprising a further step of:
inflating either or both MIN-WND and MAX-WND in response to each duplicate acknowledgement.
 4. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 3 comprising further steps of:
(5) counting a reset timer hereinafter called RsT in connection with the congestion window, and
(6) allowing the transmission of one or more packets of data into the connection if at least one of C-WND, MIN-WND and RsT permits said transmission
 5. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 3 wherein steps (3) and (4) is performed only during a portion of the connection period.
 6. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 4 wherein steps (3) and (4) is performed only during a portion of the connection period.
 7. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 3 comprising further steps of:
(7) counting a reset timer hereinafter called RsT in connection with the congestion window;

and
(8) allowing the transmission of one or more packets of data into the connection if at least one of C-WND, MAX-WND and RsT permits said transmission.

8. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 7 wherein steps (3) and (4) is performed only during a portion of the connection period.
9. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 4 wherein the connection is a TCP connection.
10. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 7 wherein the connection is a TCP connection.
11. An apparatus for sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed minimum bandwidth and a maximum permitted bandwidth, comprising:
a flow control module for controlling a flow of packets into the connection in response to acknowledged packets;
a congestion window arithmetic module for calculating a congestion window hereinafter called C-WND of the connection;
a bandwidth monitoring window arithmetic module for calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth hereinafter called MAX-WND;
control logic module for determining if the MIN-WND or MAX-WND is invoked on the connection;
a transmitter for transmitting a series of packets of data into the connection; and
a controller for allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.
12. The apparatus for sending a packet of data via a connection through a network, according to claim 11, further comprising:
a packetizing module for packetizing data to be sent into a series of packets, each having an individual sequence number, and
a clock for timing operations of various mod-

ules.

13. The apparatus for sending a packet of data via a connection through a network, according to claim 12, wherein the packet of data is a TCP packet of data and the apparatus further comprising:

a TCP protocol module for forming series of TCP packets of data, each having a sequence byte number; and
the flow control module for controlling a flow of packets into the connection in response to acknowledged sequence byte numbers of TCP packet

14. The apparatus for sending a packet of data via a connection through a network, according to claim 13, wherein the flow control module comprises further a window control module for adjusting the size of either or both MIN-WND and MAX-WND in response to each duplicate ACK.

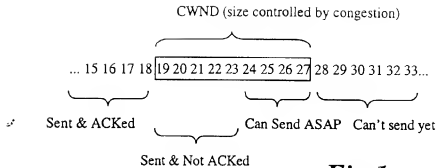


Fig 1

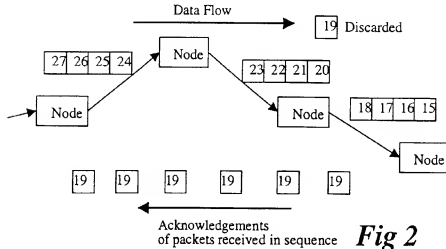
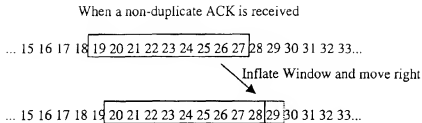
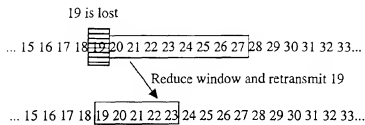
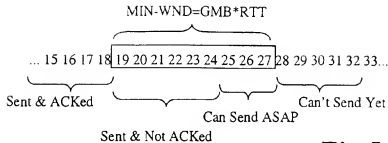
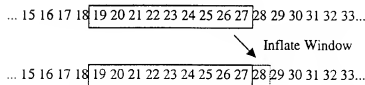


Fig 2

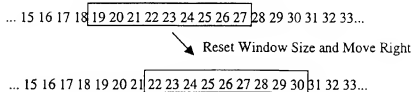
**Fig 3****Fig 4**

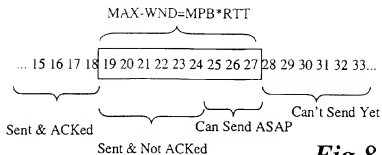
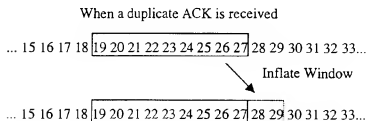
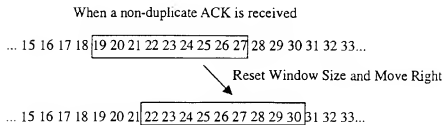
**Fig 5**

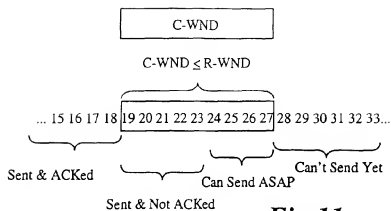
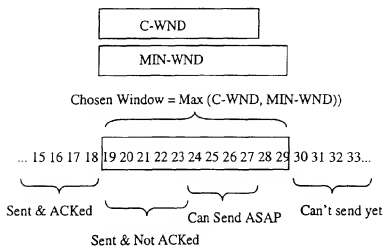
When a duplicate ACK is received

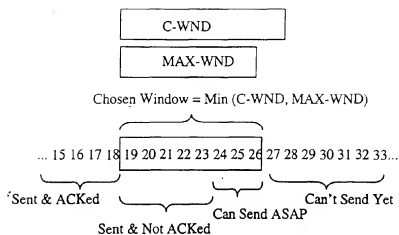
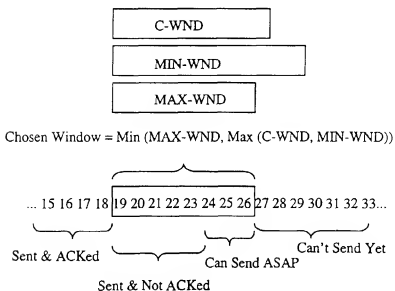
**Fig 6**

When a non-duplicate ACK is received

**Fig 7**

**Fig 8****Fig 9****Fig 10**

**Fig 11****Fig 12**

**Fig 13****Fig 14**

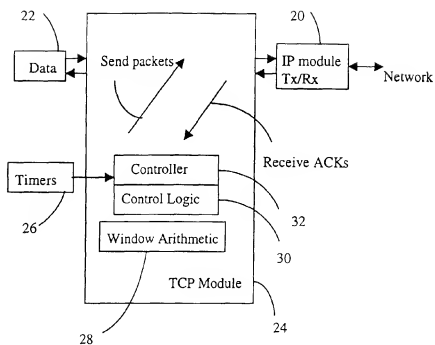


Fig 15

(12) **EUROPEAN PATENT APPLICATION**

(88) Date of publication A3:
10.05.2000 Bulletin 2000/19

(51) Int Cl⁷ **H04L 29/08, H04L 29/06,
H04L 12/56**

(43) Date of publication A2:
05.04.2000 Bulletin 2000/14

(21) Application number: **99307651.2**

(22) Date of filing: **28.09.1999**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE**
Designated Extension States:
AL LT LV MK RO SI

(72) Inventors:

- Chapman, Alan Stanley John
Kanata, Ontario K2K 1V5 (CA)
- Kung, Hsiang-Tsung
Lexington, MA 02173 (US)

(30) Priority: **30.09.1998 CA 2249152**

(74) Representative: **Dearling, Bruce Clive et al
Hepworth Lawrence Bryer & Bizley,
Merlin House,
Falconry Court,
Bakers Lane
Epping, Essex CM16 5DQ (GB)**

(71) Applicant: **Nortel Networks Corporation
Montreal, Quebec H2Y 3Y4 (CA)**

(54) **Apparatus for and method of managing bandwidth for a packet based connection**

(57) Flow control of packet based traffic by window is known. Novel modification is described which causes the flow control mechanism to reduce sending rate to some configured number rather than just reducing it by a fixed amount such as one half. The description also shows how the flow control mechanism can be constrained to a maximum rate. The configured numbers will assure that the connection can always run at a minimum rate but not more than a maximum rate. If the

guaranteed minimum bandwidth is known and the round trip time between the end points is known or has been calculated, then the sender node needs only reduce its window to that which corresponds to a sending rate equal to that configured number. In this way the protocol will still probe for extra, opportunistic bandwidth but will be able to maintain the minimum rate. In a similar way a window that corresponds to the maximum rate can be calculated and used to constrain the maximum rate of sending

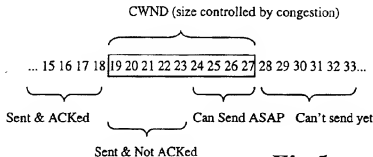


Fig 1



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 7651

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	EP 0 458 033 A (IBM) 27 November 1991 (1991-11-27) * page 5, line 53 - page 6, line 40 * * claims 1-9 *	1-14	H04L29/08 H04L29/06 H04L12/56
A	BRAKMO L S ET AL: "TCP VEGAS: NEW TECHNIQUES FOR CONGESTION DETECTION AND AVOIDANCE" COMPUTER COMMUNICATIONS REVIEW, US, ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK, vol. 24, no. 4, 1 October 1994 (1994-10-01), pages 24-35, XP000477041 ISSN: 0146-4833 * page 26, right-hand column, paragraph 3.1 - page 30, right-hand column, line 26 *	1-14	
A	"DYNAMIC COMPUTATION OF TCP MAXIMUM WINDOW SIZE FOR DIRECTLY CONNECTED HOSTS" IBM TECHNICAL DISCLOSURE BULLETIN, US, IBM CORP., NEW YORK, vol. 37, no. 4A, 1 April 1994 (1994-04-01), pages 601-607, XP000446797 ISSN: 0018-8689 * page 601, line 1 - page 604, line 44 *	1-14	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 20 March 2000	Examiner Karavassilis, N
CATEGORY OF CITED DOCUMENTS X: particularly relevant if taken alone Y: particularly relevant if combined with another document of the same category A: technology background O: non-written disclosure P: intermediate document T: theory or principle underlying the invention E: earlier patent document, but published on, or after the filing date D: document cited in the application L: document cited for other reasons S: member of the same patent family, corresponding document			

EPO FORM 1503 (12-92) (P4/C4)

(19) World Intellectual Property Organization
International Bureau(43) International Publication Date
3 May 2001 (03.05.2001)

PCT

(10) International Publication Number
WO 01/31852 A1

(51) International Patent Classification: H04L 12/28

Southampton, Hampshire SO52 9FT (GB). KFOGH, David, Bryan [GB/GB]: 25 Chichester Close, Hedge End, Southampton, Hampshire SO30 2GQ (GB).

(21) International Application Number: PCT/GB00/03979

(22) International Filing Date: 18 October 2000 (18.10.2000)

(74) Agent: ALLEN, Derek; Intellectual Property Department, Siemens Shared Services Limited, Siemens House, Oldbury, Bracknell, Berkshire RG12 8FZ (GB).

(25) Filing Language: English

(26) Publication Language: English

(81) Designated States (national): CA, JP, US.

(30) Priority Data:
9925003.7 22 October 1999 (22.10.1999) GB
9925004.5 22 October 1999 (22.10.1999) GB
0006487.3 18 March 2000 (18.03.2000) GB

(84) Designated States (regional): European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:

— With international search report.

— Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.

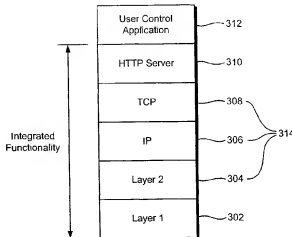
(71) Applicant (for all designated States except US): ROKE, MANOR RESEARCH LIMITED [GB/GB]: Roke Manor, Old Salisbury Lane, Romsey, Hampshire SO51 0ZN (GB).

(72) Inventors: and

(75) Inventors/Applicants (for US only): BURNETT, Alan, Mark [GB/GB]: 6 Cedar Crescent, North Baddesley,

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A FULLY INTEGRATED WEB ACTIVATED CONTROL AND MONITORING DEVICE



(57) Abstract: A web-enabled microcontroller device is provided with both web server functions (310) and generic control and monitoring functions (312). The web-enabled microcontroller device may be embedded in domestic, commercial and industrial hardware. Integrated software for remotely controlling hardware by means of the microcontroller device combines control application code (312) and HTTP server code (310). One implementation of the microcontroller device has a microprocessor coupled to a physical communications unit, a ROM and a RAM. The protocol stack (314) associated with the HTTP server may be permanently coded into the ROM or loaded into the RAM as required. In another implementation, the physical communications unit includes a digital signal processor and a wireless access unit, thereby providing a web-enabled digital wireless access device. In the wireless implementation, the processing of the wireless access physical layer (302) is performed by the digital signal processor and the higher layer processing (310, 312, 314) is performed on the microprocessor.

A FULLY INTEGRATED WEB ACTIVATED CONTROL AND MONITORING DEVICE

5 This invention relates to a fully integrated, Web activated control and monitoring device. The control device of the invention can be used for remotely controlling and monitoring functions in a physical device.

10 The Internet and World Wide Web are now ubiquitous technologies for information transfer. One prominent data networking standard for information transfer is TCP/IP (Transmission Control Protocol / Internet Protocol). The majority of home computers are now supplied 'Internet Ready' and 'Web-enabled', ensuring that many consumers are not only aware of the technology, but already have it in their homes.

15 Web servers deliver content according to a given protocol for presentation on a remote client device, using a Web browser. In the original World Wide Web (WWW) concept, the given protocol was a Hypertext Transfer Protocol (HTTP). An HTTP server delivers Hypertext Markup Language (HTML) pages to one or more users, for display by an HTTP browser. The HTML pages are static files containing text and image information. This concept has evolved to include dynamic HTML, whereby interactive pages (having radio buttons and drop down menus for example) provide an interface to other applications such as search engines, databases or online
20 dictionaries. Web servers have evolved so that in addition to supporting HTTP, the Web servers can support other information transfer protocols including a wireless application protocol (WAP), the
25

WAP being particularly appropriate for delivering content in wireless implementations.

By embedding a Web server in a target physical device, the WWW concept can be further extended to include the control and monitoring of the physical device. Examples of suitable target physical devices include video recorders, central heating systems and home security systems. The device being controlled can present a protocol-compliant interface to any network using IP (for example the public Internet, an Intranet or a home network) and can thus interface with any Web browser. The structure and content of the protocol-compliant interface presented to the Web browser is stored entirely on the controlled device.

Consider the case of an HTTP server which can be linked to a control application. The control application in turn interfaces with hardware components of the physical device. Examples of hardware components which can be controlled in this manner include: Digital to Analogue Converters (DAC); Analogue to Digital Converter (ADC); parallel Input/Output (I/O) or serial I/O devices; display devices; and device monitoring logic. Rather than simply serving fixed HTML pages, the HTTP server may deliver control and monitoring information using dynamic Web page graphical constructs, for example radio buttons or pull-down menus. Consequently, the control application provides the HTTP server with device status information in HTML format and HTML formatted menus for the setting of device control parameters. The HTTP content presented to a user may be a simulation (or exact copy) of the interface presented by the physical device being controlled.

In general, Web servers have been designed to operate on high performance, general-purpose computing platforms, and are optimised for serving content pages at high speed to a large number of users. It is known to provide HTTP servers for control applications by embedding a general-purpose computer in the physical device to be controlled or by controlling a number of local devices from a single computer. However, such an approach is clearly unsuitable for consumer applications due to the high cost of implementation.

Alternatively physical devices can be controlled by embedded microprocessors (or microcontrollers) rather than general-purpose computers; Web servers suitable for the embedded microprocessors have been produced. Typically, these Web servers are not fundamentally different in design to those used in larger scale general-purpose computing environments. Generally, the Web server software for the embedded microprocessors is smaller and supports fewer features. However, Web servers for general purpose computers and servers for the embedded microprocessors do still have the following features in common.

Firstly, both types of Web server are implemented as a separate piece of code that executes in a separate thread or process alongside the control application code.

Secondly, an external scheduling system (usually part of an operating system) determines when both the Web server software and the control application software get execution time.

Thirdly, as the control application code and the Web server code are both designed to execute separately, code needed to perform standard tasks will often be duplicated in both, adding to the overall size of the software.

Lastly, both types of Web server code incorporate code which handles differences between protocols. In the case of the HTTP, there is no single standard definition, rather there are multiple standard versions, each of which has many optional features. In addition, there are a number of non-HTTP features that are widely considered to be server options. Since a Web server is typically a separate piece of code, designed to be applied across a wide range of applications, the Web server usually contains code to support functionality that is not used by a given specific application. The implementation of a Web server in an embedded environment as described above is not optimal in a typical consumer appliance implementation using a low cost microcontroller. The memory and processing power available in typical consumer appliances are very restricted.

Ideally, the Web server software for implementation in embedded environments should include no redundant code and the control application software should be able to dictate when any support software, such as the Web server software, receives execution time. When this control application software is implemented in embedded microcontroller devices, the embedded microcontroller devices may be used to replace existing, known microcontroller devices. In addition, the cost and inconvenience of wiring is a significant barrier to home networking. The implementation of a wireless environment is thus desirable and so there is a need for Web server technology in the context of digital radio devices in order to produce a class of physical device particularly appropriate for home networking implementations.

According to the present invention, there is provided a microcontroller device for controlling at least one physical device, the microcontroller device including a microprocessor coupled to a

physical communication unit and a memory unit, wherein the memory unit stores an integrated piece of software arranged to perform a server function and a control application function and to support protocol stacks.

5 Preferably, the physical communication unit includes a UART chip. Equally preferably, the physical communication unit includes a digital signal processor and a wireless access unit. More preferably the digital signal processor is a baseband processor.

10 Preferably, the microcontroller device is embedded in a single integrated circuit.

 Preferably the memory unit includes at least one non-volatile memory unit, the memory unit advantageously, permanently storing the integrated piece of software. More preferably, the non-volatile memory is read only memory (ROM) unit.

15 Alternatively, the memory unit includes at least one volatile memory unit. The integrated software is preferably stored as an image on the at least one volatile memory unit. The at least one volatile memory unit may be a random access memory (RAM) unit.

20 Preferably, the server functions provided by the integrated piece of software comply with a communications protocol. More preferably, the communications protocol is a Hypertext Transfer Protocol and protocol stacks supported by the integrated piece of software are compatible with the Hypertext Transfer Protocol. Equally preferably, the communications protocol is a Wireless Application Protocol and
25 protocol stacks supported by the integrated piece of software are compatible with the Wireless Application Protocol.

 Preferably, there is provided a physical device controlled by the microcontroller device. More preferably, there is provided a

communications network comprising at least one physical device controlled by the microcontroller device.

According to the present invention, there is also provided a method of remotely controlling a physical device having at least one function, the method including the steps of: providing the physical device with a server engine for receiving incoming communications from a remote user; providing the physical device with a command handler arranged to produce an interpreted remote user communication by interpreting the incoming communication; providing the physical device with at least one control application arranged to receive the interpreted remote user communication and to interface with the physical device for controlling the at least one function; and controlling the at least one function in response to the at least one control application.

Preferably, a memory unit having an integrated piece of software recorded thereon is provided, wherein the integrated piece of software performs the method above.

At least one embodiment of the present invention will now be described, by way of example only, with reference to the following drawings, in which:

Figure 1 is a schematic diagram of an IP-based home network;

Figure 2A is a schematic diagram of the layout of a first web-enabled microcontroller device;

Figure 2B is a schematic diagram of the layout of a second web-enabled microcontroller device for use in a wireless network environment;

Figure 3 is a first reference model for the first web-enabled generic microcontroller device of Figure 2A;

Figure 4 is a schematic diagram of a network showing wireless and wired inter-working between devices using different physical layer media; and

Figure 5 is a schematic diagram of the integration of an HTTP server with a control application.

Throughout the following description, identical reference numerals are used to identify like parts.

Referring to Figure 1, an IP based home network 102 is coupled to a plurality of physical devices, the IP based home network 102 being in communication with the public Internet 100 via an IP gateway 104. The plurality of physical devices can include: computing devices, such as a home computer 114 or a Personal Digital Assistant (PDA) 128; home entertainment devices, such as a television 116, a video recorder 112, home audio equipment 120; domestic appliances, such as a refrigerator 124, a fax machine 126, a mobile telephone 130, a microwave oven 108 or heating and lighting facilities 118; security equipment 110; a utility meter 106, such as a gas meter or an electricity meter; and automotive monitoring equipment such as car engine management equipment 122.

Remote control of the plurality of physical devices requires that each physical device has a microcontroller arrangement for control and monitoring of the physical device, and embedded HTTP server software. The HTTP server software permits communication with a management terminal, for example the home computer 114 or the wireless web enabled PDA 128. The plurality of physical devices are managed using an HTTP browser implemented upon the management terminal in order to present information generated by the HTTP server software in the form of content pages. In order to manage the plurality

of physical devices, the management terminal requires respective IP addresses associated with each of the plurality of physical devices. The respective IP addresses can be obtained by an automatic discovery process, for example the Bluetooth 'Service Discovery Protocol'.

5 Although reference has been made to "Bluetooth" it should be appreciated that other wireless access techniques, for example, Infrared Data Association (IrDA) or Shared Wireless Access Protocol (SWAP), can be used to reduce the cost and complexity of network wiring.

10 In a first embodiment of the invention (Figure 2A), a web-enabled microcontroller arrangement 200 includes a physical communications unit 202 coupled to a microprocessor 210 via a first data bus 212, a non-volatile memory unit 204 (e.g. ROM, EEPROM, EPROM or flash memory) and a volatile memory unit 206 (e.g. RAM or random access flash) also being coupled to the first data bus 212.

15 The microcontroller arrangement 200 is suitable for connection to a network for communications with remote devices coupled to the network by means of the physical communications unit 202, for example, a Universal Asynchronous Receiver/Transmitter chip (UART).

20 Referring to Figure 3, the web-enabled microcontroller arrangement 200 implements an integrated protocol stack 314 providing a data link layer (Layer 2) 304, an IP layer 306 and a TCP layer 308, the IP layer 306 and the TCP layer 308 providing transmission and networking layers. The integrated protocol stack 314

25 is overlaid upon a physical layer 302 (Layer 1). An HTTP layer 310 is overlaid upon the TCP layer 308, the HTTP layer 310 being implemented by HTTP server software. The HTTP server software, the integrated protocol stack 314 and user control application software

312 all execute on the embedded microprocessor 210. The physical layer 302, the data link layer 304, the IP layer 306, the TCP layer 308 and the HTTP layer 310 are either coded permanently into the non-volatile memory unit 204 (for example, mask programmed onto ROM) or provided as an image which may be loaded by a software developer into the volatile memory unit 206. In the latter case, the software developer is free to generate control application software specific to the hardware of the controlled device and linking directly to the HTTP server software.

The code build process incorporates the building of both control application code and HTTP server code to give a single piece of web-enabled software. This approach gives a number of advantages in terms of overall size and complexity of the web-enabled software, and in terms of utilisation of available processing power.

Firstly, the HTTP server software presents a generic user interface with content dependent only on the controlled device itself. The HTTP server software uses a networking standard (TCP/IP) which allows a device to be monitored or controlled from anywhere in the world with no translation whatsoever of the content presented by the HTTP server. The device can be controlled from a known platform; for example, a workstation, the home personal computer 114 or the PDA 128.

Secondly, the HTTP server code and the control application code are combined to form a single piece of software in which the HTTP server code only executes when called by the control application code. By structuring the HTTP server code so that it periodically returns thread mastery to the control application code throughout the processing of an incoming HTTP communication, and by providing

parameters to configure this processing both dynamically and as part of the code build, the application is given a high degree of control over the available processing power. When active, the HTTP server can be seen as making use of spare capacity not used for the control application.

Thirdly, since the HTTP server code is part of an implementation specific code build process, the web-enabled control application software includes code to support any and all desirable HTTP server features. By the use of standard code building techniques and suitable structuring of the HTTP server code, any HTTP server code supporting functionality that is optional, yet not relevant for the specific implementation for the physical device, can simply be excluded from the final software as part of the code build process for that implementation.

Finally, as the control application code and HTTP server code form part of the same piece of web-enabled software, they can both make use of the same sections of code to perform standard tasks rather than duplicate these sections in each case.

In a second embodiment of the present invention (Figure 2B), the microcontroller arrangement 200 differs from that of the first embodiment (Figure 2A) by the physical communication unit 202 having an embedded DSP 214 and a radio frequency (RF) transceiver device 216, for transmitting and receiving signals.

In the context of the reference model of Figure 3, the physical layer 302 is supported by the transceiver unit 216 and the embedded DSP 214. Signals received by the transceiver unit 216 are processed by the embedded DSP 214, the embedded DSP 214 can also support the (baseband) data link layer 304 (layer 2). Although specific

apparatus have been described for the full or partial support of layers 1 and 2 of the reference model, it should be appreciated that other hardware known in the art can be used to support the layers 1 and 2.

Some functionality of the data link layer 304, the IP layer 306,
5 the TCP layer 308 and the HTTP layer 210 is provided by the embedded microprocessor 210. Typical wireless access technologies that use this device architecture include: Digital Enhanced Cordless Telecommunications (DECT), Global System for Mobile communications (GSM), Bluetooth, Universal Mobile
10 Telecommunications System (UMTS), IrDA or SWAP.

It should be understood that the embedded microprocessor 210 can be instructed to perform the operations of the DSP 214 in addition to Layer 2 and control processing, thus the DSP 214 is not absolutely essential. Alternatively, logic circuitry can be provided to perform
15 some or all tasks of the DSP 214 and the embedded microprocessor 210.

Referring to Figure 4, the wired and wireless inter-working between devices of Figure 1 can be seen in more detail. Layers 1 and 2 of the wired part of the network differ from layers 1 and 2 of the
20 wireless part of the network. However, the IP layer 306, the TCP layer 308 and the HTTP server layer 310 are identical, making inter-working straightforward.

In addition to the above described applications, physical devices comprising the microcontroller arrangement 200 can be used for
25 diagnostic and configuration functions by a remote service centre. For example, a user can have difficulty configuring one of the physical devices or the physical device may appear to be faulty. The user can connect the physical device to the remote service centre via the public

Internet 100, where skilled service personnel can run diagnostics remotely or assist the user in the configuration of the physical device. Additionally, the home entertainment equipment incorporating the embedded Web server can include specialised configuration and diagnostic functions not accessible to the user. The configuration and diagnostic functions, inaccessible to the user, can be accessed by a remote customer service centre via the public Internet 100.

The home security systems 110 can also have additional remote management features. A security company can use embedded HTTP server technology to provide additional management services to a client. The HTTP server software can be used to control and monitor audio and video surveillance equipment, and also to control building infrastructure functions, for example, heat and power systems.

In another application, the utility meter 106 can be read and controlled remotely by both a customer and a utility provider via the wired public Internet 100 or a wireless network in order to save on collection/reading costs and to provide a common user interface.

The HTTP server software embedded in the car engine management system 122 can provide a valuable diagnostic tool. The engine management system 122 collects performance and service interval information, which can be interrogated locally by a garage, or remotely over the public Internet 100 by a manufacturer's service centre or a breakdown service. Information gathered can be used for a number of purposes including: by the garage to identify items requiring adjustment/replacement during service; by the garage to identify a fault after breakdown; by the breakdown service to identify a fault before dispatching roadside assistance; and by the manufacturer to gather performance information over the life of a vehicle.

In order to exclude unauthorised users from the home IP network 102 connected to the public Internet 100 where authorised users are relatively unskilled, a firewalling technique is employed. In general, the firewalling technique restricts access to local networks from the unauthorised users. The firewalling technique is implemented at the gateway 104 between the public Internet 100 and the home IP network 102. Access to the home IP network 102 is restricted by applying security measures known in the art including filtering of incoming packets on combinations of identification numbers including source IP addresses, destination IP addresses, TCP/IP port numbers and UDP (User Datagram Protocol) port numbers.

In a wired network the firewalling function is performed by an IP access router known in the art acting as a Local Area Network (LAN) switch/router for the controlled devices connected to it. However, in a wireless network, it is possible that the gateway is only an access point, and performs no switching or routing functions for the devices in communication with the wireless network. The firewalling technique is still required for the wireless network, but no physical media switching is needed. Consequently, the gateway 104 comprises a firewalling device implementing the firewalling technique and the protocol relay function in order to provide an access point to the public Internet 100 to a home IP network 102. In this case, the firewalling device is just another web-enabled physical device and can be managed, configured and diagnosed remotely, with expert help as required.

Referring to Figure 5, the control application 506 is capable of relaying commands to a hardware control module 530 and receives responses from the hardware control module 530. The HTTP server

software includes an HTTP server engine 502 and a plurality of command handlers 504. Each command handler 504 is a piece of application specific code that the HTTP server engine 502 uses to handle a respective application specific aspect of an HTTP communication for the control application 506.

The control application 506 registers each command handler 504 with the HTTP server engine 502. The HTTP server engine 502 then calls one of the plurality of command handler 504 appropriate for handling a respective application specific aspect of an HTTP communication at various stages of the HTTP communication. The registration process associates a textual name with each command handler 504, thus allowing the appropriate command handler 504 to be specified within the HTTP communication (not shown). The HTTP server engine 502 looks for textual names of command handlers 504 within HTTP communications and maps each textual name found, using the list of registered command handlers, to an appropriate corresponding command handler 504.

Furthermore, the command handlers 504 are specific to the hardware ultimately being controlled through the hardware control module 530. Any parameters and context specific content passed by the HTTP server engine 502 to the control application 506 via a given command handler 504 are interpreted by the given command handler 504, the command handler 504 acting as an interface between the control application 506 and the Web server engine 502. In response to the parameters and context specific content delivered via the command handler 504, the control application 506 performs whatever action is appropriate given the nature of the HTTP communication (not shown). The command handlers 504 also interpret the results of actions by the

control application 506 and act as the interface back to the HTTP server engine 502, for example the command handlers 504 identify errors and/or produce parameters and context specific content for the HTTP server engine 502 to format into an outgoing HTTP communication (not shown). The HTTP server engine 502 handles all aspects of the HTTP connection, from receiving and sending of HTTP communications, to parsing and formatting the parameters and context specific content.

The HTTP server engine 502 also handles sequencing, by calling an appropriate command handler 504 at various points throughout the process of handling an HTTP communication. The sequencing is, however, still under the overall control of the control application 506, due to other features of the HTTP server software. The control application 506 can, either as part of the build process or dynamically for each HTTP communication, specify parameters, for instance, how much data may be sent or received over the TCP/IP connection before returning execution to the control application 506. Additionally, in the sequence of events that are required to handle an HTTP communication, there are fixed points where the control application 506 can define whether the HTTP server engine 502 continues automatically with the next event of the sequence or returns execution to the control application 506 after each event. The HTTP server engine 502 therefore is only allowed execution time when called by the control application 506. The amount of execution time allocated at each call to the HTTP server engine 502 can thus be configured, as can the number of simultaneous HTTP communications that can be handled.

The user can still interact locally with the hardware of the physical device through a local user interface 540. Local user communications (not shown) are handled directly by the control application 506, the control application 506 acting as an interface to the hardware control module 530. Feedback from the hardware control module 530 returns to the user interface 540 via the control application 506.

It will be understood that, although the foregoing description is concerned with HTTP server code, other protocols can be adopted in place of hypertext transfer protocol. Most notably the Wireless Application Protocol (WAP) is suitable for the implementation of a Web server in a wireless environment. Suitable alternative protocols within which the invention can be applied include file transfer protocol (FTP), Session Initiation Protocol (SIP), Service Location Protocol (SLP), telnet and secure HTTP (SHTTP). Suitable protocol stacks for the physical layer 302 and the data link layer (Layer 2) 304 include: X.25, AppleTalk, Ethernet and asynchronous transfer mode (ATM).

Claims:

1. A microcontroller device for controlling at least one physical device, the microcontroller device including a microprocessor coupled
5 to a physical communication unit and a memory unit, wherein the memory unit stores an integrated piece of software arranged to perform a server function and a control application function and to support protocol stacks.
- 10 2. A microcontroller device according to Claim 1, wherein the physical communication unit includes a UART chip.
3. A microcontroller device according to Claim 1, wherein the physical communication unit includes a digital signal processor and a
15 wireless access unit.
4. A microcontroller device according to Claim 3, wherein the digital signal processor is a baseband processor.
- 20 5. A microcontroller device according to Claims 1, 2, 3 or 4, wherein the microcontroller device is embedded in a single integrated circuit.
- 25 6. A microcontroller device according to any one of Claims 1 to 5, wherein the memory unit includes at least one non-volatile memory unit.

7. A microcontroller device according to Claim 6, wherein the at least one non-volatile memory unit permanently stores the integrated piece of software.

5 8. A microcontroller device according to Claims 6 or 7, wherein the at least one non-volatile memory unit is a read only memory unit.

9. A microcontroller device according to any one of Claims 1 to 6, wherein the memory unit includes at least one volatile memory unit.

10 10. A microcontroller device according to Claim 9, wherein the integrated piece of software is stored as an image on the at least one volatile memory unit.

15 11. A microcontroller device according to Claims 9 or 10, wherein the at least one volatile memory unit is a random access memory unit.

12. A microcontroller device according to any one of the preceding claims, wherein the server functions provided by the integrated piece of
20 software comply with a communications protocol.

13. A microcontroller device according to Claim 12, wherein the communications protocol is a Hypertext Transfer Protocol and protocol stacks supported by the integrated piece of software are compatible
25 with the Hypertext Transfer Protocol.

14. A microcontroller device according to Claim 12, wherein the communications protocol is a Wireless Application Protocol and

protocol stacks supported by the integrated piece of software are compatible with the Wireless Application Protocol.

15. A physical device controlled by a microcontroller device as
5 claimed in any one of the preceding claims.

16. A communications network comprising at least one physical device according to Claim 15.

10 17. A method of remotely controlling a physical device having at least one function, the method including the steps of:

providing the physical device with a server engine for receiving incoming communications from a remote user;

15 providing the physical device with a command handler arranged to produce an interpreted remote user communication by interpreting the incoming communication;

providing the physical device with at least one control application arranged to receive the interpreted remote user communication and to interface with the physical device for controlling
20 the at least one function; and

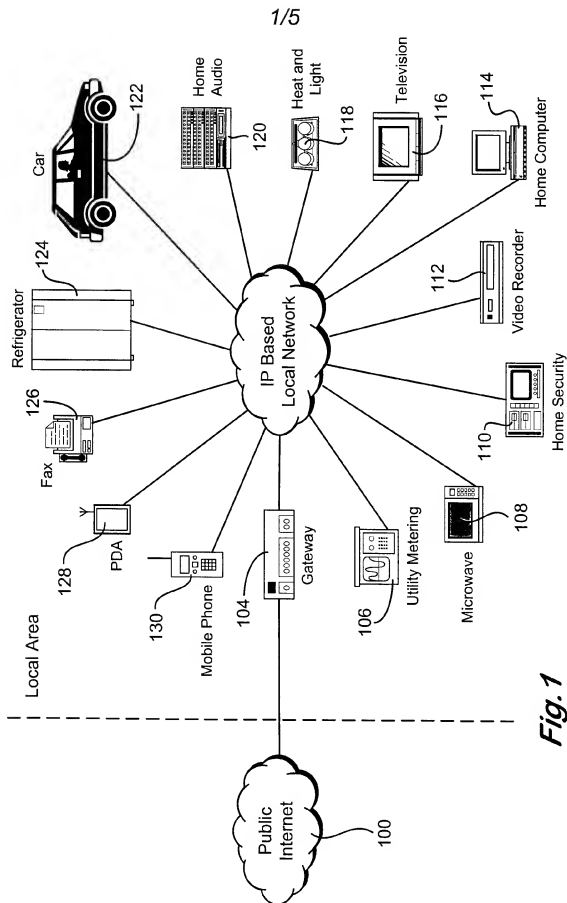
controlling the at least one function in response to the at least one control application.

18. A memory unit having an integrated piece of software recorded
25 thereon, wherein the integrated piece of software performs the method as claimed in Claim 17.

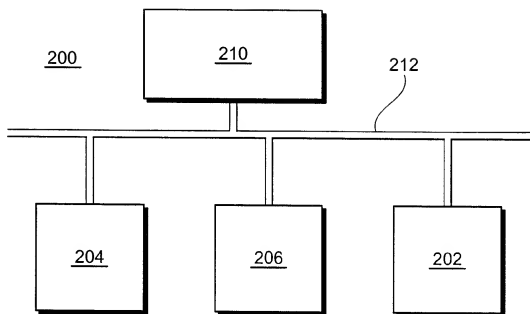
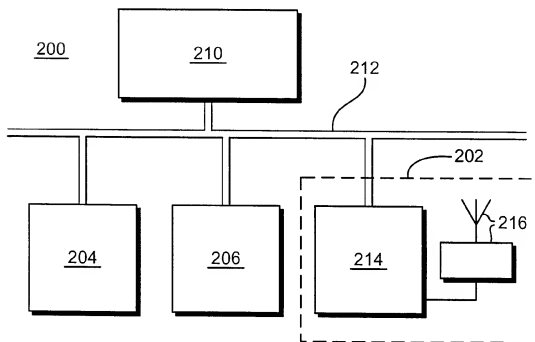
19. A microcontroller device substantially as hereinbefore described with reference to the accompanying drawings.

20. A network of microcontroller devices substantially as
5 hereinbefore described with reference to the accompanying drawings.

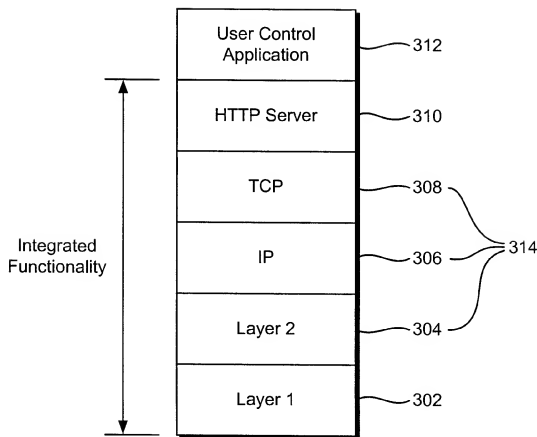
21. A physical device substantially as hereinbefore described with reference to the accompanying drawings.



2/5

**Fig.2A****Fig.2B**

3/5

***Fig. 3***

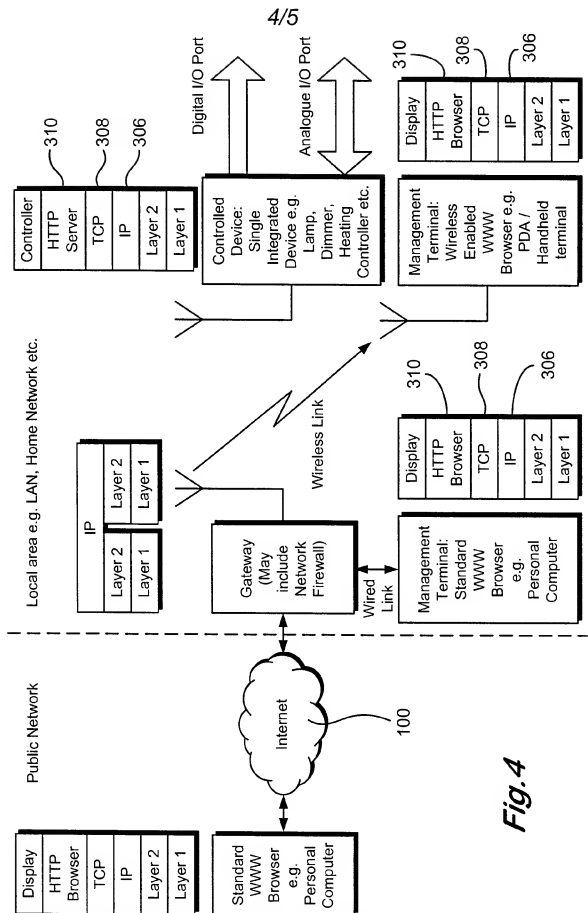
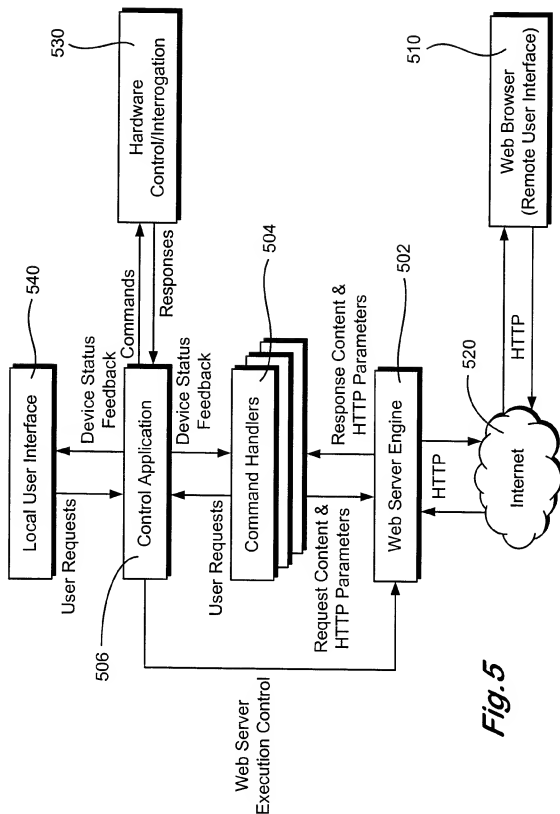


Fig. 4

5/5

*Fig. 5*

INTERNATIONAL SEARCH REPORT

International Application No.

PCT/GB 00/03979

A. CLASSIFICATION OF SUBJECT MATTER

IPC 7 H04L12/28

According to International Patent Classification (IPC) into both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	<p>PEISEL B: "DESIGNING THE NEXT STEP IN INTERNET APPLIANCES" ELECTRONIC DESIGN, US, PENTON PUBLISHING, CLEVELAND, OH, vol. 46, no. 7, 23 March 1998 (1998-03-23), page 50, 52, 56 XP000780455 ISSN: 0013-4872 page 50, left-hand column, line 1 - page 56, left-hand column, line 58</p> <p>---</p> <p>---</p>	<p>1, 5-8, 12, 13, 15-21</p>

<input checked="" type="checkbox"/> Further documents are listed in the continuation of box C.	<input checked="" type="checkbox"/> Patent family members are listed in annex.
<p>* Special categories of cited documents:</p> <p>"A" document defining the general state of the art which is not considered to be of particular relevance</p> <p>"E" earlier document but published on or after the international filing date</p> <p>"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)</p> <p>"O" document referring to an oral disclosure, use, exhibition or other means</p> <p>"P" document published prior to the international filing date but later than the priority date claimed</p> <p>"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention</p> <p>"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone</p> <p>"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.</p> <p>"&" document member of the same patent family</p>	
Date of the actual completion of the international search	Date of mailing of the international search report
28 February 2001	06/03/2001
Name and mailing address of the ISA European Patent Office, P.B. 5616 Patentkan 2 NL - 2260 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo.nl, Fax. (+31-70) 340-3016	Authorized officer Pham, P

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/GB 00/03979

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	LAWTON G: "DAWN OF THE INTERNET APPLIANCE" COMPUTER,US,IEEE COMPUTER SOCIETY, LONG BEACH., CA, US, vol. 30, no. 10, 1 October 1997 (1997-10-01), page 16,18 XP000738076 ISSN: 0018-9162 the whole document -----	1,9, 11-13, 15-21
X	WO 99 46746 A (ABB POWER T & D CO) 16 September 1999 (1999-09-16)	1,12,13, 15-21
A	page 3, line 24 -page 7, line 3 page 8, line 20 -page 9, line 3 -----	3

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/GB 00/03979

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9946746 A	16-09-1999	AU 6453498 A	27-09-1999
		EP 1062648 A	27-12-2000
		NO 20004486 A	08-11-2000



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L 29/00	A2	(11) International Publication Number: WO 00/67443 (43) International Publication Date: 9 November 2000 (09.11.00)
(21) International Application Number: PCT/NO00/00144 (22) International Filing Date: 28 April 2000 (28.04.00) (30) Priority Data: 19992125 30 April 1999 (30.04.99) NO (71) Applicant (for all designated States except US): TELEFONAK- TIEBOLAGET LM ERICSSON [SE/SE]; S-126 25 Stock- holm (SE). (72) Inventor; and (75) Inventor/Applicant (for US only): NILSEN, Børge [NO/NO]; Lørenveien 34, N-0585 Oslo (NO). (74) Agent: OSLO PATENTKONTOR AS; Postboks 7007 M, N-0306 Oslo (NO).		(81) Designated States: AE, AG, AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), DM, DZ, EE, EE (Utility model), ES, FI, FI (Utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KR (Utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG). Published <i>Without international search report and to be republished upon receipt of that report.</i>
(54) Title: ADAPTION OF SERVICES IN A TELEPHONE NETWORK (57) Abstract <p>The present invention relates to an arrangement for improving the service architecture for a compound network, comprising several types of access, as well as comprising parallel service nodes/networks for respective access technologies, and for the purpose of making customer specific adaptations to the service layer more flexible and allowing for a more cost-effective support of access specific protocols and service, it is according to the present invention suggested that said arrangement comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures.</p>		

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MY	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	NL	Netherlands	US	United States of America
CA	Canada	IT	Italy	NO	Norway	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NZ	New Zealand	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	PL	Poland	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PT	Portugal		
CN	China	KR	Republic of Korea	RO	Romania		
CU	Cuba	KZ	Kazakhstan	RU	Russian Federation		
CZ	Czech Republic	LC	Saint Lucia	SD	Sudan		
DE	Germany	LI	Liechtenstein	SE	Sweden		
DK	Denmark	LK	Sri Lanka	SG	Singapore		
EE	Estonia	LR	Liberia				

ADAPTION OF SERVICES IN A TELEPHONE NETWORK***FIELD OF THE INVENTION***

- 5 The present invention relates to an arrangement for improving the service architecture for a compound telephone network, comprising several types of access/protocols as well as comprising parallel service nodes.
- 10 In other words, the present invention finds its application in the field of H.323 and service control.

BACKGROUND OF THE INVENTION

- 15 The present invention has been developed in connection with problems encountered when making service specific adaptations to the requirements of the individual customer in a network.
- 20 The problem is that the respective service logic is highly integrated into the switching logic of the core network. This means that new service logic and service adaptations may require changes in the core switching functions. This again makes it very hard to make customer specific adaptations to the service layer.
- 25

For example, big changes in the IN services can not be made without updating the switches in the network, i.e. even if the control in reality is between the handset and node 1 in

30 the network, there is "transport equipment" that also has to be updated as service control uses the same control data/mechanisms/paths as these transport nodes.

KNOWN SOLUTIONS AND PROBLEMS WITH THESE

35

As explained above the main problem with traditional networks is that the service control protocols are integrated with lower layer functions such as call and media control.

This binds the service control plane to lower layer functions such as basic switching functions and introduce system couplings that make customisation of service control expensive.

5

The service control and services are thus often tied to given access protocols. This often leads to building of parallel service networks with minor differences in protocols and services. Each service network then solves the service issues for one specific type of net. This typically results in costly and ineffective service development frameworks that are inflexible, costly and hard to maintain.

15 For the IP telephone networks as defined by the H.323 standard, the service control protocol is defined by the H.450 standards-suite. This defines an in-band service control protocol that is carried within the H.225.0 call control plane (a defined subset of Q.931). This protocol defines a set of ASN.1 service control messages that are used for invoking and controlling services. The problems with this approach is summarized as:

25 Introduction of new services requires updates of H.450 messages and decoding logic in the gatekeeper. This slows down introduction of service logic as it requires both a standardisation process and updates to the switch control plane.

30 Adaptation of service control to vendor specific control messages/logic becomes impossible (or costly) as it relates to the switching core.

35 Integration and interworking with messaging protocols becomes heavier as it requires more transcoding of messages. The user-user messages and user-service messages are carried within the same messages and framing. In order to

identify user-service messages, all messages need to be analysed - not only those addressed for services. The protocol is ASN.1 encoded and does not easily integrate with MIME encoded messaging services.

5

OBJECTS OF THE INVENTION

An object of the present invention is to provide an arrangement, by which a cost effective and adaptive service architecture for a compound network comprising several types of access, can be implemented in a far more expedient and versatile manner.

Another object of the present invention is to provide an arrangement, by which the service networks can more easily be integrated and developed.

Still another object of the present invention is to provide an arrangement whereby the service architecture and access technologies are made more flexible and more easy to maintain.

BRIEF SUMMARY OF THE INVENTION

The above objects are achieved in an arrangement as stated in the preamble, which according to the present invention is characterized in that said comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures, said open service control protocol is adapted for removing the coupling between the access/service technology and the switching logic in the core network.

In other words, the invention aims at customization of service control protocols by allowing the service control to be specialised independently from the other control functions such as call control and media control.

The invention proposes to use an open service control protocol that allows for a more cost-effective support of access specific protocols and services while also allowing
5 the respective access networks to share the same service nodes and service architectures. The solution also aims at removing the coupling between the access/service technology and the switching logic in the core network. The proposed solution is based on H.323 being extended with HTTP for the
10 service control.

Further features and advantages will appear from the following description taken in conjunction with the enclosed drawings, as well as from the enclosed patent claims.

15

BRIEF DISCLOSURES OF THE DRAWINGS

Figure 1 is a schematical layout illustrating multiple access type and service node architecture.

20

Figure 2 is a schematical layout illustrating the general principle of the present invention, comprising composed single service node with plugin architecture, giving homogeneous architecture with access adapters.

25

Figure 3 is a simplified block diagram illustrating a reference model according to the present invention.

30

Figure 4 is a schematical layout illustrating a service node structure according to an embodiment of the present invention.

35

Figure 5 is a schematical layout illustrating an example of usage according to the present invention.

DETAILED DESCRIPTION OF EMBODIMENTS

In Figure 1 there is in a schematical layout illustrated a prior art multiple access type and service node architecture.
5

In order to elevate the problems related to the service architecture for such a compound network comprising several types of access, as well as comprising parallel service
10 nodes/networks for respective access technologies, it is according to the present invention suggested to use an open service control protocol, as this will be discussed in further detail with reference to Figure 2.

15 Figure 2 illustrates a homogeneous service architecture with access adapters, according to the present invention, by which the open service control protocol can be implemented, so as to allow for a more cost effective support of access specific protocols and services while also allowing
20 active access networks to share the same service nodes and service architectures.

With the suggested solution which can be embedded in the general layout according to Figure 2, it is also possible
25 to remove the coupling between the access/service technology and the switching logic in the core network.

The proposed solution is based on H.323 being extended with HTTP for service control.

30 More specifically, the proposed solution replaces the H.450 standards suites with the more open and extendable HTTP protocol. The solution also makes use of the feature set of HTTP to add the required flexibility. Among these features
35 are found:

HTTP Tunnelling

- The tunnelling feature refers to the use of the HTTP transport layer protocol for carrying other data protocols (in which case the HTTP headers carry information about what kind of payload type/protocol that is being carried).

Server Side Plugin

- 10 The plugin approach represents the server side equivalent of browser plugins where 3rd party plugins (objects/functions) can be added dynamically. The invocation of a plugin is controlled through the content-type field or through selections/filters on the given path. The binding between the plugin and the invocation criteria is set through configuration.

Servlet Functions

- 20 The servlet approach relates to servlet objects that implements CGI like functions, but may add persistency over sessions and is being object oriented.

DESCRIPTION OF SOLUTION

- 25 The invention disclosure relates to an H.323 based telephone network where clients makes phone calls through a central call-/control processing switch called a gatekeeper. The gatekeeper performs call-/control processing functions such as charging, routing and resource control and may also activate call related services on a service node according to the call states and the user profiles. When the client and the gatekeeper talks different languages/dialects, an access node is added in between to perform the required gateway functions.

In Figure 3 there is illustrated a simplified network reference model.

This invention disclosure proposes to replace the H.450 based service control protocol between the access nodes and the gatekeeper/service node with an HTTP based protocol.

- 5 This means that service configuration- and control messages are being HTTP encoded by the access nodes and decoded, analysed and executed by the service node. This again means that there is no normalisation of the service protocols in the access nodes and that the mapping from the access specific service data to the service node languages are being
10 performed by service node plugins/servlets.

- The service node represents a set of software processes being capable of executing phone services and interacting
15 with the gatekeeper. The service node thus provides a set of service functions and offers a programming API for service execution and control. The service node does also provide an HTTP server that supports HTTP tunnelling, servlets and server side plugins. Through the use of this HTTP
20 server it is possible to write a plugin or servlet that interacts with the programming API in order to control the service execution. An example of this could be a plugin that translates DTMF codes to API method calls, e.g. DTMF '*23*1*1530#' may translate to API method 'userIs-
25 BusyTo(15.30)'.

In Figure 4 there is given an illustration of the service node architecture.

- 30 In order to provide acceptable service availability the service node and the HTTP server will need to support installations of new plugins/servlets in runtime. Further, the architecture needs to be such that faulty plugins, servlets or sessions does not impair the operation of the
35 service node.

The described architecture and feature set supports access specific service control messages as well as customer spe-

cific adaptation of the service network and the service control protocols, though within the limits of the feature set of the service API. This is illustrated through the following two examples.

5

In order to add an access specific service control protocol such as QSIG, the vendor would need to write an access node and a service plugin/servlet.

10 The QSIG access-node would translate the call- and connection control messages into the H.323 format, but would tunnel the QSIG messages inside HTTP messages and address these to the service node.

15 A QSIG plugin/servlet would be written and installed on the HTTP server of the service node. The logic of this plugin/servlet would translate the QSIG messages into method calls (and capability sets) in the service API. When a QSIG service control message is sent from a PBX, the
20 access node will wrap the QSIG message into an HTTP frame and send it to the service node. The HTTP server on the service node will receive the package, detect that the format is something called QSIG, look up in its configuration data and activate the correct plugin/servlet for QSIG. This
25 plugin/servlet will analyse the QSIG message, make method calls in the API and return the appropriate QSIG encoded response.

When new features are added to the service node and the
30 service API, the updates can be provided to the access specific parts through updates of the plugins/servlets, i.e. there are no updates required in the access nodes.

In order to add a provider specific service control protocol e.g. based on GSM-SMS, the provider would need to write
35 a plugin/servlet that translates the GSM-SMS messages into method calls over the service API. The procedures are as defined above, but in this case an external 3rd part can do

provider specific customisation to the service network without being tied up to new deliveries of the core system. In Figure 5 it is illustrated the GSM-SMS example alongside a default option.

5

ADVANTAGES

Added Flexibility

- 10 The service control protocol is more flexible in terms of supporting different service control data formats/encoding standards. For each new encoding standard, a new plugin needs to be encoded.

Simplicity

- 15 The service control protocol becomes more flexible in that it is simpler to add new service control messages and supporting these. It becomes simpler to debug the system, to
20 secure the message transport (cf. SSL) and to get the data through firewalls and proxies.

Customisation

- 25 The solution allows the service provider to add provider specific service control messages independent from the system solution provider. This means that a provider can add new control messages for decoding these independent from the system provider (e.g. add a new SMS message and update
30 the plugin for decoding this).

Performance

- The messages are being addressed towards the correct recipient, meaning that the gatekeeper does not need to analyse all messages (incl. user-user msg.) in order to pick
35 up the user-service data.

BROADENING**1) Integration with Messaging Applications**

- 5 The HTTP service control format follows the MIME encoding standard that is used by SMTP, NTP and S/MIME messaging applications. It is expected that it should be possible to integrate this service control with these messaging applications.

10

2) Support for Notification Services

- The principle can be extended to allow the application server to issue HTTP messages/notifications to the clients (e.g. when the client registers). This can for example be used for notifying the user about new e-mail messages in the in-box.

20

3) Extensions for Supporting the SIP Protocol

The SIP protocol builds on using the HTTP protocol and can probably be integrated into the system solution relatively simple if the application server supports call-from-the-blue services.

25

4) Terminal (gateway) to Terminal (gateway) Service Control

- If two terminals (or their gateways) want to exchange service control/data they could exchange this service control/data on a language that they have agreed on. The respective entities (terminals or gateways) can also dynamically download transcoder servlets/plugins from a central depository upon need.

- 35 This could for example be used when user A on his PC is sending user B on a GSM terminal an email message. The GSM gateway decides that email is not understood and retrieves some transcoder for handling this email. The choice of

transcoder can be selected according to user preferences, previous user behaviour, network or operator criteria. Examples of transcoders here could be:

- 5 • Transcoder from email to GSM-SMS message
- Transcoder from email to voice rendering
- Transcoder from email to WAP

5) Access Control based on Service Control Plugin

10

The access to transcoder functions (servlets/plugins) can be controlled according to subscription profiles, user locations and other metrics of the system. Further more can the invoked transcoder function apply access control on the

15 specific information elements of the service control/data protocols. This could e.g. be used to control when and from where a given service is used and what kind of service data that is legal in the given context. An example could be to filter on the contents of a GSM-SMS message to ensure that

20 no pornographic data is being transmitted. (The transcoder would in this case act as an application layer firewall.)

APPENDIX**Terminology**

ITU-H.323	A family of ASN.1 encoded protocols defining message formats, encoding standards and call state sequences of multimedia conferences on an Internet protocol infrastructure.
ITU-H.225.0	A subset of the H.323 standards suite being based on Q.931 and defining call control messages, encoding standards and call-state sequences.
ITU-H.450	A suite of ASN.1 standards defining service control protocols to be used for service control in an H.323 network. The H.450 messages are being carried within H.225.0 messages.
ITU-Q.931	Telephony standard for call control that defines call control messages, encoding standards and call-state sequences.
ASN.1	Abstract Syntax Notation Number 1
HTTP	A formal data structure definition language A MIME (ascii) encoded protocol for transport of world-wide-web data. The protocol is open for tunnelling of other protocols.
CGI	Common Gateway Interface A script language used for customisation of web page contents
API	Application Programming Interface
DTMF	Dual Tone Multiple Frequency
QSIG	A service control protocol used by PBX
PBX	Private Branch Exchange
GSM	Global System for Mobile Communication A widely employed standard for mobile communication
SMS	Short Message Service Messaging service protocol employed within GSM
SSL	Secure Socket Layer Security protocol employed for Transport Layer

	Security
MIME	Multipart Information Message Entity Protocol encoding format based on ascii characters
SIP	Session Initiation Protocol IP Telephony protocol based on HTTP
SMTP	Simple Mail Transfer Protocol Protocol for transport/exchange of email mes- sages
NTTP	Network News Transfer Protocol Protocol for transport/exchange of news mes- sages
S/MIME	Secure MIME
WAP	Wireless Access Protocol A web protocol for mobile devices (i.e. 'a- kind-of' HTTP for mobile handsets)

P a t e n t c l a i m s

1. Arrangement for providing an improved service architecture for a compound telephone network,
5 c h a r c t e r i z e d i n that said arrangement comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures,
10 said open service control protocol is adapted for removing the coupling between the access/service technology and the switching logic in the core network.
2. Arrangement as claimed in claim 1,
15 c h a r c t e r i z e d i n that said open service control protocol is based on H.323 standard for communication across Internet Protocol (IP) based networks, said H.323 standard being extended with HTTP (Hyper Text Transport Protocol) for the service control.
- 20 3. Arrangement as claimed in claim 1 or 2,
c h a r c t e r i z e d i n that said H.323 standard with extended HTTP protocol is adapted to enhance or replace the H.450 suite of protocols, the adaptation also
25 making use of the feature set of HTTP to add the required flexibility.
4. Arrangement as claimed in claim 3,
c h a r c t e r i z e d i n that said features of the
30 HTTP may include:
- HTTP tunnelling
 - Service Side Plugin
 - Servlet Functions
- 35 5. Arrangement as claimed in claim 3 or 4,
c h a r c t e r i z e d i n that between the access nodes and the gatekeeper/service node there is introduced

an HTTP based protocol, which entails that the service configuration and control messages are being HTTP encoded by the access nodes and decoded, analysed and executed by the service node.

5

6. Arrangement as claimed in claim 5,
c h a r c t e r i z e d i n that no normalisation of
the service protocols in the access node has to be per-
formed, and that the mapping from the access specific serv-
10 ice data to the service node languages are being performed
by service node plugins/servlets.

7. Arrangement as claimed in claim 6,
c h a r c t e r i z e d i n that the service node rep-
15 represents a set of software processes being capable of exe-
cuting phone services and interacting with the gatekeeper,
the service node thus providing a set of service functions
and offering a programming API for service execution and
control.

20

8. Arrangement as claimed in claim 1,
c h a r c t e r i z e d i n that said network includes
a service node which also provides an HTTP server that sup-
ports HTTP tunnelling, servlets and server side plugins,
25 the use of this HTTP server making it possible to write a
plugin or servlet that interacts with the programming API
in order to control the service execution.

9. Arrangement as claimed in claim 8,
30 c h a r c t e r i z e d i n that the arrangement com-
prises a plugin/servlet that translates from access spe-
cific service control to generic service API method calls.

10. Arrangement as claimed in claim 9,
35 c h a r c t e r i z e d i n that the access node is
adapted to tunnel the access specific service control data
to the plugin/servlet by use of HTTP, which plugin/servlet

then transcodes this access specific service control to said generic service API method calls.

11. Arrangement as claimed in claim 9 or 10,
5 c h a r a c t e r i z e d i n t h a t a s e r v e r s i d e
plugin/servlet can be installed and updated in run-time.

12. Arrangement as claimed in claim 11,
c h a r a c t e r i z e d i n t h a t s a i d s e r v e r s i d e
10 plugin/servlet can be provided by 3rd parties.

13. Arrangement as claimed in any of the claims 9-12,
c h a r a c t e r i z e d i n t h a t s a i d s e r v e r a u t o m a t i c a l l y
selects correct plugin/servlet according to configured rules in the server and the type of service control
15 data being signalled, the type of data being signalled is indicated by the HTTP protocol.

14. Arrangement as claimed in any of the claims 9-13,
20 c h a r a c t e r i z e d i n t h a t t h e p l u g i n / s e r v l e t
will format the return codes/states from the generic API calls to access specific return codes/states and return these using the HTTP protocol.

15. Arrangement as claimed in claim 14,
c h a r a c t e r i z e d i n t h a t t h e r e s p e c t i v e e n t i t i e s
involved (terminal or gateway) can dynamically download transcoder servlets/plugins from a central repository
upon need.

30 16. Arrangement as claimed in claims 15,
c h a r a c t e r i z e d i n t h a t a c c e s s t o t r a n s c o d e r
functions (servlets/plugins) can be controlled according to subscriber profiles, user locations and other metrics of
35 the system.

17. Arrangement as claimed in claims 15-16,
c h a r a c t e r i z e d i n t h a t t h e i n v o k e d

transcoder function can be arranged to apply access control to the specific information elements of the service control/data protocols.

Fig-1: Access Specific Service Architecture / multiple Service Nodes

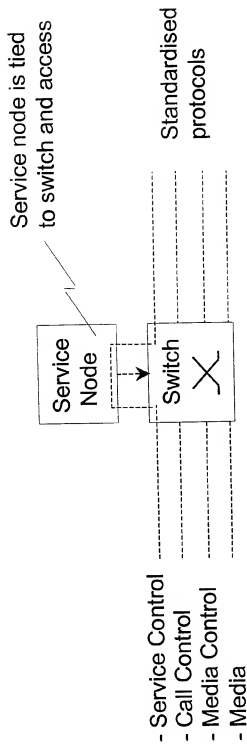


Fig-2: Homogenous Service Architecture with Access Adaptors

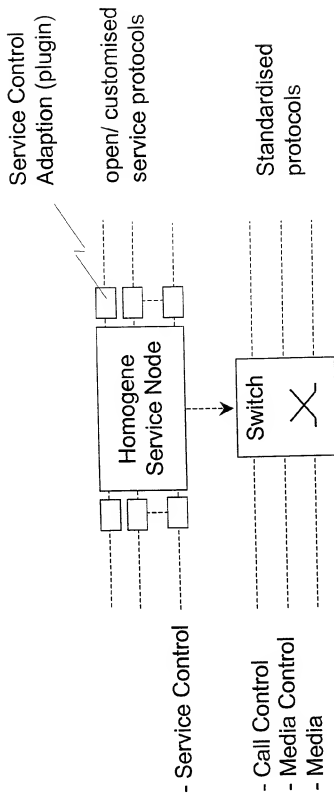


Fig-3: Simplified Reference Model

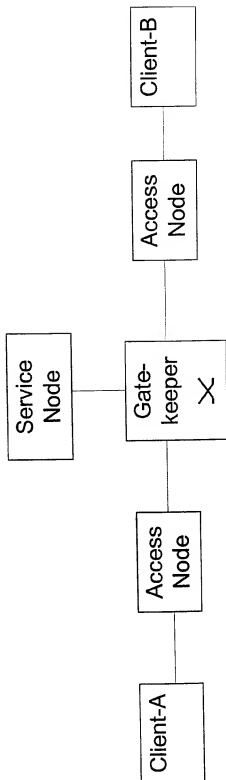
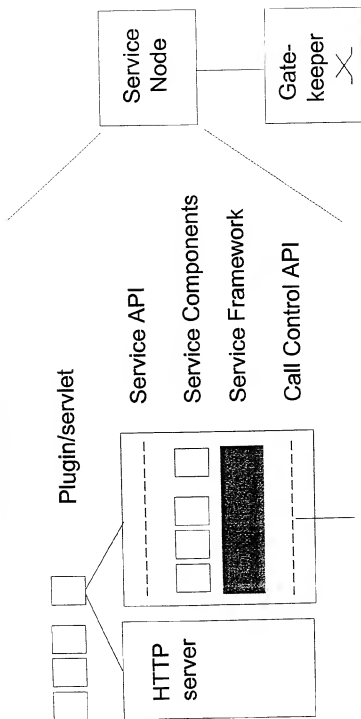
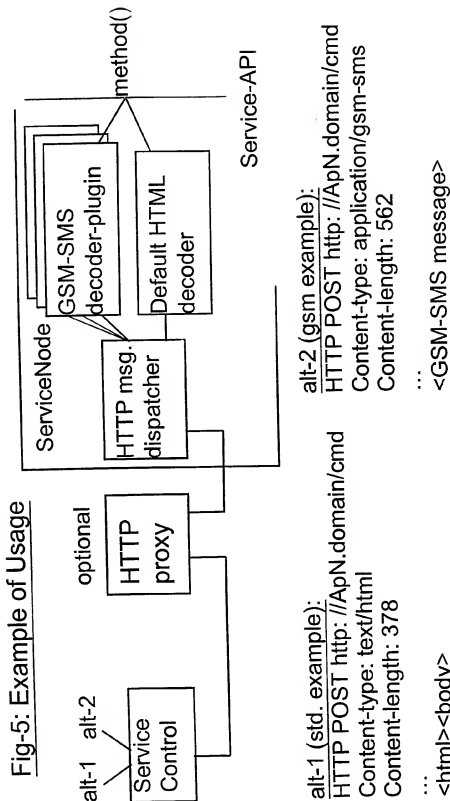


Fig-4: Service Node Structure





(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
9 November 2000 (09.11.2000)

PCT

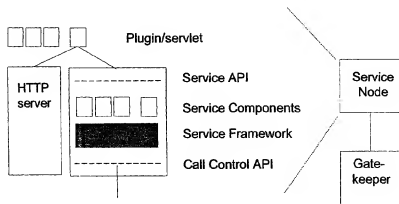
(10) International Publication Number
WO 00/67443 A3

- (51) International Patent Classification⁷: **H04L 12/64** (81) Designated States (*national*): AE, AG, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KR (utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.
- (21) International Application Number: PCT/NO00/00144
- (22) International Filing Date: 28 April 2000 (28.04.2000)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data: 19992125 30 April 1999 (30.04.1999) NO (84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, I.U, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).
- (71) Applicant (*for all designated States except US*): TELEFONAKTIEBOLAGET LM ERICSSON [SE/SE]; S-126 25 Stockholm (SE).
- (72) Inventor; and
- (75) Inventor/Applicant (*for US only*): NILSEN, Berge [NO/NO]; Lørenveien 34, N-0585 Oslo (NO). Published: — With international search report.
- (74) Agent: OSLO PATENTKONTOR AS; Postboks 7007 M, N-0306 Oslo (NO).
- (88) Date of publication of the international search report: 25 January 2001

[Continued on next page]

(54) Title: ADAPTION OF SERVICES IN A TELEPHONE NETWORK

Service Node Structure



(57) Abstract: The present invention relates to an arrangement for improving the service architecture for a compound network, comprising several types of access, as well as comprising parallel service nodes/networks for respective access technologies, and for the purpose of making customer specific adaptations to the service layer more flexible and allowing for a more cost-effective support of access specific protocols and service, it is according to the present invention suggested that said arrangement comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures.



For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

International Application No.

PCT/NO 00/00144

A. CLASSIFICATION OF SUBJECT MATTER

IPC 7 H04L12/64

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L H04Q H04M

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No
E	EP 1 003 343 A (NORTEL NETWORKS CORP) 24 May 2000 (2000-05-24) the whole document ---	1
P,A	DANIELE RIZZETTO ET AL: "A voice over IP Service Architecture for Integrated Communications" IEEE INTERNET COMPUTING, vol. 3, May 1999 (1999-05) - June 1999 (1999-06), pages 53-62, XP002901209 ISSN 1089-7801 the whole document ---	1
P,A	EP 0 996 270 A (NORTEL NETWORKS CORP) 26 April 2000 (2000-04-26) the whole document ---	1
	--- -/--	

☒ Further documents are listed in the continuation of box C

☒ Patent family members are listed in annex

* Special categories of cited documents :

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- *Z* document member of the same patent family

Date of the actual completion of the international search

29 August 2000

Date of mailing of the international search report

31. 10. 2000

Name and mailing address of the ISA
European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax. (+31-70) 340-3016

Authorized officer

Mans Marklund

INTERNATIONAL SEARCH REPORT

International Application No.

PCT/NO 00/00144

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,A	EP 0 989 705 A (AT & T CORP) 29 March 2000 (2000-03-29) the whole document -----	1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No.

PCT/NO 00/00144

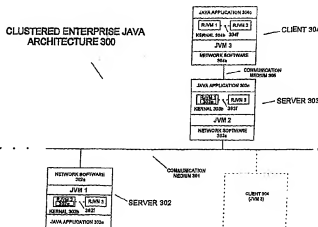
Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 1003343 A	24-05-2000	NONE	
EP 0996270 A	26-04-2000	NONE	
EP 0989705 A	29-03-2000	CN 1250996 A	19-04-2000



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶: G06F 15/16	A1	(11) International Publication Number: WO 00/28431 (43) International Publication Date: 18 May 2000 (18.05.00)
(21) International Application Number: PCT/US99/24561 (22) International Filing Date: 21 October 1999 (21.10.99) (30) Priority Data: 60/107,167 5 November 1998 (05.11.98) US 09/405,318 23 September 1999 (23.09.99) US (71) Applicant: BEA SYSTEMS, INC. [US/US]; 2315 North First Street, San Jose, CA 95131 (US). (72) Inventors: JACOBS, Dean, B.; 1747 Madera Street, Berkeley, CA 94707 (US); LANGEN, Anno, R.; 2220-d Sacramento Street, Berkeley, CA 94702 (US). (74) Agents: MEYER, Sheldon, R. et al.; Fliesler Dubb Meyer & Lovejoy LLP, Suite 400, Four Embarcadero Center, San Francisco, CA 94111-4156 (US).		(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SI, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG). Published <i>With international search report.</i> <i>Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>

(54) Title: CLUSTERED ENTERPRISE JAVA™ HAVING A MESSAGE PASSING KERNEL IN A DISTRIBUTED PROCESSING SYSTEM

**(57) Abstract**

A clustered enterprise Java™ distributed processing system is provided. The distributed processing system includes a first and a second computer coupled to a communication medium. The first computer includes a Java™ virtual machine (JVM) and kernel software layer for transferring messages, including a remote Java™ virtual machine (RJVM). The second computer includes a JVM and a kernel software layer having a RJVM. Messages are passed from a RJVM to the JVM in one computer to the JVM and RJVM in the second computer. Messages may be forwarded through an intermediate server or rerouted after a network reconfiguration. Each computer includes a Smart stub having a replica handler, including a load balancing software component and a failover software component. Each computer includes a duplicated service naming tree for storing a pool of Smart stubs at a node. The computers may be programmed in a stateless, stateless factory, or a stateful programming model. The clustered enterprise Java™ distributed processing system allows for enhanced scalability and fault tolerance.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

**CLUSTERED ENTERPRISE JAVA™ HAVING A MESSAGE PASSING
KERNEL IN A DISTRIBUTED PROCESSING SYSTEM**

Field of the Invention

5 The present invention relates to distributed processing systems
and, in particular, computer software in distributed processing
systems.

Cross Reference to Related Applications

10 This application claims the benefit of U.S. Provisional
Application No. 60/107,167, filed November 5, 1998.

The following copending U.S. patent applications are assigned
to the assignee of the present application, and their disclosures are
incorporated herein by reference:

15 (A) Ser. No. Not Yet Known [Attorney Docket No. BEAS1029]
filed Not Yet Known by Dean B. Jacobs and Eric M. Halpern, and
entitled, "A SMART STUB OR ENTERPRISE JAVA™ BEAN IN A
DISTRIBUTED PROCESSING SYSTEM";

20 (B) Ser. No. Not Yet Known [Attorney Docket No. BEAS1030]
filed Not Yet Known by Dean B. Jacobs and Eric M. Halpern, and
entitled, "A DUPLICATED NAMING SERVICE IN A DISTRIBUTED
PROCESSING SYSTEM"; and

 (C) Ser. No. Not Yet Known [Attorney Docket No. BEAS1031]
filed Not Yet Known by Dean B. Jacobs and Anno R. Langen, and
originally entitled, "CLUSTERED ENTERPRISE JAVA™ IN A SECURE
DISTRIBUTED PROCESSING SYSTEM".

25 **Background of the Invention**

There are several types of distributed processing systems.
Generally, a distributed processing system includes a plurality of

processing devices, such as two computers coupled to a communication medium. Communication mediums may include wired mediums, wireless mediums, or combinations thereof, such as an Ethernet local area network or a cellular network. In a distributed
5 processing system, at least one processing device may transfer information on the communication medium to another processing device.

Client/server architecture 110 illustrated in Fig. 1a is one type of distributed processing system. Client/server architecture 110
10 includes at least two processing devices, illustrated as client 105 and application server 103. Additional clients may also be coupled to communication medium 104, such as client 108.

Typically, server 103 hosts business logic and/or coordinates transactions in providing a service to another processing device, such
15 as client 103 and/or client 108. Application server 103 is typically programmed with software for providing a service. The software may be programmed using a variety of programming models, such as Enterprise Java™ Bean ("EJB") 100b as illustrated in Figs. 1a-b. The service may include, for example, retrieving and transferring data
20 from a database, providing an image and/or calculating an equation. For example, server 103 may retrieve data from database 101a in persistent storage 101 over communication medium 102 in response to a request from client 105. Application server 103 then may transfer the requested data over communication medium 104 to
25 client 105.

A client is a processing device which utilizes a service from a server and may request the service. Often a user 106 interacts with client 106 and may cause client 105 to request service over a communication medium 104 from application server 103. A client

often handles direct interactions with end users, such as accepting requests and displaying results.

A variety of different types of software may be used to program application server 103 and/or client 105. One programming language is the Java™ programming language. Java™ application object code is loaded into a Java™ virtual machine ("JVM"). A JVM is a program loaded onto a processing device which emulates a particular machine or processing device. More information on the Java™ programming language may be obtained at <http://www.javasoft.com>, which is incorporated by reference herein.

Fig. 1b illustrates several Java™ Enterprise Application Programming Interfaces ("APIs") 100 that allow Java™ application code to remain independent from underlying transaction systems, data-bases and network infrastructure. Java™ Enterprise APIs 100 include, for example, remote method invocation ("RMI") 100a, EJBs 100b, and Java™ Naming and Directory Interface (JNDI) 100c.

RMI 100a is a distributed programming model often used in peer-to-peer architecture described below. In particular, a set of classes and interfaces enables one Java™ object to call the public method of another Java™ object running on a different JVM.

An instance of EJB 100b is typically used in a client/server architecture described above. An instance of EJB 100b is a software component or a reusable pre-built piece of encapsulated application code that can be combined with other components. Typically, an instance of EJB 100b contains business logic. An EJB 100b instance stored on server 103 typically manages persistence, transactions, concurrency, threading, and security.

JNDI 100c provides directory and naming functions to Java™ software applications.

Client/server architecture 110 has many disadvantages. First, architecture 110 does not scale well because server 103 has to handle many connections. In other words, the number of clients which may be added to server 103 is limited. In addition, adding
5 twice as many processing devices (clients) does not necessarily provide you with twice as much performance. Second, it is difficult to maintain application code on clients 105 and 108. Third, architecture 110 is susceptible to system failures or a single point of failure. If server 101 fails and a backup is not available, client 105
10 will not be able to obtain the service.

Fig. 1c illustrates a multi-tier architecture 160. Clients 151, 152 manage direct interactions with end users, accepting requests and display results. Application server 153 hosts the application code, coordinates communications, synchronizations, and
15 transactions. Database server 154 and portable storage device 155 provides durable transactional management of the data.

Multi-tier architecture 160 has similar client/server architecture 110 disadvantages described above.

Fig. 2 illustrates peer-to-peer architecture 214. Processing
20 devices 216, 217 and 218 are coupled to communication medium 213. Processing devices 216, 217, and 218 include network software 210a, 210b, and 210c for communicating over medium 213. Typically, each processing device in a peer-to-peer architecture has similar processing capabilities and applications. Examples of
25 peer-to-peer program models include Common Object Request Broker Architecture ("CORBA") and Distributed Object Component Model ("DCOM") architecture.

In a platform specific distributed processing system, each processing device may run the same operating system. This allows

the use of proprietary hardware, such as shared disks, multi-tailed disks, and high speed interconnects, for communicating between processing devices. Examples of platform-specific distributed processing systems include IBM® Corporation's S/390® Parallel Sysplex®, Compaq's Tandem Division Himalaya servers, Compaq's Digital Equipment Corporation™ (DEC™) Division OpenVMS™ Cluster software, and Microsoft® Corporation Windows NT® cluster services (Wolfpack).

Fig. 2b illustrates a transaction processing (TP) architecture 220. In particular, TP architecture 220 illustrates a BEA® Systems, Inc. TUXEDO® architecture. TP monitor 224 is coupled to processing devices ATM 221, PC 222, and TP monitor 223 by communication medium 280, 281, and 282, respectively. ATM 221 may be an automated teller machine, PC 222 may be a personal computer, and TP monitor 223 may be another transaction processor monitor. TP monitor 224 is coupled to back-end servers 225, 226, and 227 by communication mediums 283, 284, and 285. Server 225 is coupled to persistent storage device 287, storing database 289, by communication medium 286. TP monitor 224 includes a workflow controller 224a for routing service requests from processing devices, such as ATM 221, PC 222, or TP monitor 223, to various servers such as server 225, 226 and 227. Work flow controller 224a enables (1) workload balancing between servers, (2) limited scalability or allowing for additional servers and/or clients, (3) fault tolerance of redundant backend servers (or a service request may be sent by a workflow controller to a server which has not failed), and (4) session concentration to limit the number of simultaneous connections to back-end servers. Examples of other transaction processing architectures include IBM® Corporation's CICS® ,

Compaq's Tandem Division Pathway/Ford/TS, Compaq's DEC™ ACMS, and Transarc Corporation's Encina.

TP architecture 220 also has many disadvantages. First, a failure of a single processing device or TP monitor 224 may render the network inoperable. Second, the scalability or number of processing devices (both servers and clients) coupled to TP monitor 224 may be limited by TP monitor 224 hardware and software. Third, flexibility in routing a client request to a server is limited. For example, if communication medium 280 is inoperable, but communication medium 290 becomes available, ATM 221 typically may not request service directly from server 225 over communication medium 290 and must access TP monitor 224. Fourth, a client typically does not know the state of a back-end server or other processing device. Fifth, no industry standard software or APIs are used for load balancing. And sixth, a client typically may not select a particular server even if the client has relevant information which would enable efficient service.

Therefore, it is desirable to provide a distributed processing system and, in particular, distributed processing system software that has the advantages of the prior art distributed processing systems without the inherent disadvantages. The software should allow for industry standard APIs which are typically used in either client/server, multi-tier, or peer-to-peer distributed processing systems. The software should support a variety of computer programming models. Further, the software should enable (1) enhanced fault tolerance, (2) efficient scalability, (3) effective load balancing, and (4) session concentration control. The improved computer software should allow for rerouting or network reconfiguration. Also, the computer

software should allow for the determination of the state of a processing device.

SUMMARY OF THE INVENTION

5 An improved distributed processing system is provided and, in particular, computer software for a distributed processing system is provided. The computer software improves the fault tolerance of the distributed processing system as well as enables efficient scalability. The computer software allows for efficient load balancing and session concentration. The computer software supports rerouting or 10 reconfiguration of a computer network. The computer software supports a variety of computer programming models and allows for the use of industry standard APIs that are used in both client/server and peer-to-peer distributed processing architectures. The computer software enables a determination of the state of a server or other 15 processing device. The computer software also supports message forwarding under a variety of circumstances, including a security model.

According to one aspect of the present invention, a distributed processing system comprises a communication medium coupled to a 20 first processing device and a second processing device. The first processing device includes a first software program emulating a processing device ("JVM1") including a first kernel software layer having a data structure ("RJVM1"). The second processing device includes a first software program emulating a processing device 25 ("JVM2") including a first kernel software layer having a data structure ("RJVM2"). A message from the first processing device is transferred to the second processing device through the first kernel software layer and the first software program in the first processing

device to the first kernel software layer and the first software program in the second processing device.

According to another aspect of the present invention, the first software program in the first processing device is a Java™ virtual machine ("JVM") and the data structure in the first processing device is a remote Java™ virtual machine ("RJVM"). Similarly, the first software program in the second processing device is a JVM and the data structure in the second processing device is a RJVM. The RJVM in the second processing device corresponds to the JVM in the first processing device.

According to another aspect of the present invention, the RJVM in the first processing device includes a socket manager software component, a thread manager software component, a message routing software component, a message compression software component, and/or a peer-gone detection software component.

According to another aspect of the present invention, the first processing device communicates with the second processing device using a protocol selected from the group consisting of Transmission Control Protocol ("TCP"), Secure Sockets Layer ("SSL"), Hypertext Transport Protocol ("HTTP") tunneling, and Internet InterORB Protocol ("IIOP") tunneling.

According to another aspect of the present invention, the first processing device includes memory storage for a Java™ application.

According to another aspect of the present invention, the first processing device is a peer of the second processing device. Also, the first processing device is a server and the second processing device is a client.

According to another aspect of the present invention, a second communication medium is coupled to the second processing device. A third processing device is coupled to the second communication medium. The third processing device includes a first software
5 program emulating a processing device ("JVM3"), including a kernel software layer having a first data structure ("RJVM1"), and a second data structure ("RJVM2").

According to still another aspect of the present invention, the first processing device includes a stub having a replica-handler
10 software component. The replica-handler software component includes a load balancing software component and a failover software component.

According to another aspect of the present invention, the first processing device includes an Enterprise Java™ Bean object.

15 According to still another aspect of the present invention, the first processing device includes a naming tree having a pool of stubs stored at a node of the tree and the second processing device includes a duplicate of the naming tree.

According to still another aspect of the present invention, the first processing device includes an application program coded in a
20 stateless program model and the application program includes a stateless session bean.

According to still another aspect of the present invention, the first processing device includes an application program coded in a
25 stateless factory program model and the application program includes a stateful session bean.

According to still another aspect of the present invention, the first processing device includes an application program coded in a

stateful program model and the application program includes an entity session bean.

According to still another aspect of the present invention, an article of manufacture including an information storage medium is provided. The article of manufacture comprises a first set of digital information for transferring a message from a RJVM in a first processing device to a RJVM in a second processing device.

According to another aspect of the present invention, the article of manufacture comprises a first set of digital information, including a stub having a load balancing software program for selecting a service provider from a plurality of service providers.

According to another aspect of the present invention, the stub has a failover software component for removing a failed service provider from the plurality of service providers.

According to another aspect of the present invention, the load balancing software component selects a service provider based on an affinity for a particular service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider in a round robin manner.

According to another aspect of the present invention, the load balancing software component randomly selects a service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the load of each service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the data type requested.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the closest physical service provider.

5 According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon a time period in which each service provider responds.

10 According to another aspect of the present invention, the article of manufacture comprises a first set of digital information, including an Enterprise Java™ Bean object for selecting a service provider from a plurality of service providers.

 According to another aspect of the present invention, a stub is stored in a processing device in a distributed processing system. The
15 stub includes a method comprising the steps of obtaining a list of service providers and selecting a service provider from the list of service providers.

 According to another aspect of the present invention, the method further includes removing a failed service provider from the
20 list of service providers.

 According to still another aspect of the present invention, an apparatus comprises a communication medium coupled to a first processing device and a second processing device. The first processing device stores a naming tree including a remote method
25 invocation ("RMI") stub for accessing a service provider. The second processing device has a duplicate naming tree and the service provider.

According to another aspect of the present invention, the naming tree has a node including a service pool of current service providers.

5 According to another aspect of the present invention, the service pool includes a stub.

According to another aspect of the present invention, a distributed processing system comprises a first computer coupled to a second computer. The first computer has a naming tree, including a remote invocation stub for accessing a service provider. The
10 second computer has a replicated naming tree and the service provider.

According to another aspect of the present invention, a distributed processing system comprising a first processing device coupled to a second processing device is provided. The first
15 processing device has a JVM and a first kernel software layer including a first RJVM. The second processing device includes a first JVM and a first kernel software layer including a second RJVM. A message may be transferred from the first processing device to the second processing device when there is not a socket available
20 between the first JVM and the second JVM.

According to another aspect of the present invention, the first processing device is running under an applet security model, behind a firewall or is a client, and the second processing device is also a client.

25 Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

Brief Description of the Figures

Fig. 1a illustrates a prior art client/server architecture;

Fig. 1b illustrates a prior art Java™ enterprise APIs;

Fig. 1c illustrates a multi-tier architecture;

5 Fig. 2a illustrates a prior art peer-to-peer architecture;

Fig. 2b illustrates a prior art transaction processing architecture;

Fig. 3a illustrates a simplified software block diagram of an embodiment of the present invention;

10 Fig. 3b illustrates a simplified software block diagram of the kernel illustrated in Fig. 3a;

Fig. 3c illustrates a clustered enterprise Java™ architecture;

Fig. 4 illustrates a clustered enterprise Java™ naming service architecture;

15 Fig. 5a illustrates a smart stub architecture;

Fig. 5b illustrates an EJB object architecture;

Fig. 6a is a control flow chart illustrating a load balancing method;

20 Figs. 6b-g are control flow charts illustrating load balancing methods;

Fig. 7 is a control flow chart illustrating a failover method;

Fig. 8 illustrates hardware and software components of a client/server in the clustered enterprise Java™ architecture shown in Figs. 3-5.

25 The invention will be better understood with reference to the drawings and detailed description below. In the drawings, like reference numerals indicate like components.

DETAILED DESCRIPTION

I. Clustered Enterprise Java™ Distributed Processing System

A. Clustered Enterprise Java™ Software Architecture

Fig. 3a illustrates a simplified block diagram 380 of the software layers in a processing device of a clustered enterprise Java™ system, according to an embodiment of the present invention. A detailed description of a clustered enterprise Java™ distributed processing system is described below. The first layer of software includes a communication medium software driver 351 for transferring and receiving information on a communication medium, such as an ethernet local area network. An operating system 310 including a transmission control protocol ("TCP") software component 353 and internet protocol ("IP") software component 352 are upper software layers for retrieving and sending packages or blocks of information in a particular format. An "upper" software layer is generally defined as a software component which utilizes or accesses one or more "lower" software layers or software components. A JVM 354 is then implemented. A kernel 355 having a remote Java™ virtual machine 356 is then layered on JVM 354. Kernel 355, described in detail below, is used to transfer messages between processing devices in a clustered enterprise Java™ distributed processing system. Remote method invocation 357 and enterprise Java™ bean 358 are upper software layers of kernel 355. EJB 358 is a container for a variety of Java™ applications.

Fig. 3b illustrates a detailed view of kernel 355 illustrated in Fig. 3a. Kernel 355 includes a socket manager component 363, thread manager 364 component, and RJVM 356. RJVM 356 is a data structure including message routing software component 360, message compression software component 361 including

abbreviation table 161c, and peer-gone detection software component 362. RJVM 356 and thread manager component 364 interact with socket manager component 363 to transfer information between processing devices.

5 **B. Distributed Processing System**

Fig. 3 illustrates a simplified block diagram of a clustered enterprise Java™ distributed processing system 300. Processing devices are coupled to communication medium 301. Communication medium 301 may be a wired and/or wireless communication medium or combination thereof. In an embodiment, communication medium 301 is a local-area-network (LAN). In an alternate embodiment, communication medium 301 is a world-area-network (WAN) such as the Internet or World Wide Web. In still another embodiment, communication medium 301 is both a LAN and a WAN.

15 A variety of different types of processing devices may be coupled to communication medium 301. In an embodiment, a processing device may be a general purpose computer 100 as illustrated in Fig. 8 and described below. One of ordinary skill in the art would understand that Fig. 8 and the below description describes one particular type of processing device where multiple other types of processing devices with a different software and hardware configurations could be utilized in accordance with an embodiment of the present invention. In an alternate embodiment, a processing device may be a printer, handheld computer, laptop computer, 20 scanner, cellular telephone, pager, or equivalent thereof.

25 Fig. 3c illustrates an embodiment of the present invention in which servers 302 and 303 are coupled to communication medium 301. Server 303 is also coupled to communication medium 305

which may have similar embodiments as described above in regard to communication medium 301. Client 304 is also coupled to communication medium 305. In an alternate embodiment, client 304 may be coupled to communication medium 301 as illustrated by the dashed line and box in Fig. 3c. It should be understood that in alternate embodiments, server 302 is (1) both a client and a server, or (2) a client. Similarly, Fig. 3 illustrates an embodiment in which three processing devices are shown wherein other embodiments of the present invention include multiple other processing devices or communication mediums as illustrated by the ellipses.

Server 302 transfers information over communication medium 301 to server 303 by using network software 302a and network software 303a, respectively. In an embodiment, network software 302a, 303a, and 304a include communication medium software driver 351, Transmission Control Protocol software 353, and Internet Protocol software 352 ("TCP/IP"). Client 304 also includes network software 304a for transferring information to server 303 over communication medium 305. Network software 303a in server 303 is also used to transfer information to client 304 by way of communication medium 305.

According to an embodiment of the present invention, each processing device in clustered enterprise Java™ architecture 300 includes a message-passing kernel 355 that supports both multi-tier and peer-to-peer functionality. A kernel is a software program used to provide fundamental services to other software programs on a processing device.

In particular, server 302, server 303, and client 304 have kernels 302b, 303b, and 304b, respectively. In particular, in order for two JVMs to interact, whether they are clients or servers, each

JVM constructs an RJVM representing the other. Messages are sent from the upper layer on one side, through a corresponding RJVM, across the communication medium, through the peer RJVM, and delivered to the upper layer on the other side. In various embodiments, messages can be transferred using a variety of different protocols, including, but not limited to, Transmission Control Protocol/Internet Protocol ("TCP/IP"), Secure Sockets Layer ("SSL"), Hypertext Transport Protocol ("HTTP") tunneling, and Internet InterORB Protocol ("IIOP") tunneling, and combinations thereof. The RJVMs and socket managers create and maintain the sockets underlying these protocols and share them between all objects in the upper layers. A socket is a logical location representing a terminal between processing devices in a distributed processing system. The kernel maintains a pool of execute threads and thread manager software component 364 multiplexes the threads between socket reading and request execution. A thread is a sequence of executing program code segments or functions.

For example, server 302 includes JVM1 and Java™ application 302c. Server 302 also includes a RJVM2 representing the JVM2 of server 303. If a message is to be sent from server 302 to server 303, the message is sent through RJVM2 in server 302 to RJVM1 in server 303.

C. Message Forwarding

Clustered enterprise Java™ network 300 is able to forward a message through an intermediate server. This functionality is important if a client requests a service from a back-end server through a front-end gateway. For example, a message from server 302 (client 302) and, in particular, JVM1 may be forwarded to client

304 (back-end server 304) or JVM3 through server 303 (front-end gateway) or JVM2. This functionality is important in controlling session concentration or how many connections are established between a server and various clients.

- 5 Further, message forwarding may be used in circumstances where a socket cannot be created between two JVMs. For example, a sender of a message is running under the applet security model which does not allow for a socket to be created to the original server. A detailed description of the applet security model is provided at
10 <http://www.javasoft.com>, which is incorporated herein by reference. Another example includes when the receiver of the message is behind a firewall. Also, as described below, message forwarding is applicable if the sender is a client and the receiver is a client and thus does not accept incoming sockets.
- 15 For example, if a message is sent from server 302 to client 304, the message would have to be routed through server 303. In particular, a message handoff, as illustrated by 302f, between RJVM3 (representing client 304) would be made to RJVM2 (representing server 303) in server 302. The message would be
20 transferred using sockets 302e between RJVM2 in server 302 and RJVM1 in server 303. The message would then be handed off, as illustrated by dashed line 303f, from RJVM1 to RJVM3 in server 303. The message would then be passed between sockets of RJVM3 in server 303 and RJVM2 in client 304. The message then would be
25 passed, as illustrated by the dashed line 304f, from RJVM2 in client 304 to RJVM1 in client 304.

D. Rerouting

An RJVM in client/server is able to switch communication paths or communication mediums to other RJVMs at any time. For example, if client 304 creates a direct socket to server 302, server 302 is able to start using the socket instead of message forwarding through server 303. This embodiment is illustrated by a dashed line and box representing client 304. In an embodiment, the use of transferring messages by RJVMs ensures reliable, in-order message delivery after the occurrence of a network reconfiguration. For example, if client 304 was reconfigured to communication medium 301 instead of communication medium 305 as illustrated in Fig. 3. In an alternate embodiment, messages may not be delivered in order.

An RJVM performs several end-to-end operations that are carried through routing. First, an RJVM is responsible for detecting when a respective client/server has unexpectedly died. In an embodiment, peer-gone selection software component 362, as illustrated in Fig. 3b, is responsible for this function. In an embodiment, an RJVM sends a heartbeat message to other clients/servers when no other message has been sent in a predetermined time period. If the client/server does not receive a heartbeat message in the predetermined count time, a failed client/server which should have sent the heartbeat, is detected. In an embodiment, a failed client/server is detected by connection timeouts or if no messages have been sent by the failed client/server in a predetermined amount of time. In still another embodiment, a failed socket indicates a failed server/client.

Second, during message serialization, RJVMs, in particular, message compression software 360, abbreviate commonly transmitted data values to reduce message size. To accomplish this,

each JVM/RJVM pair maintains matching abbreviation tables. For example, JVM1 includes an abbreviation table and RJVM1 includes a matching abbreviation table. During message forwarding between an intermediate server, the body of a message is not deserialized on the intermediate server in route.

E. Multi-tier/Peer-to-Peer Functionality

Clustered enterprise Java™ architecture 300 allows for multi-tier and peer-to-peer programming.

Clustered enterprise Java™ architecture 300 supports an explicit syntax for client/server programming consistent with a multi-tier distributed processing architecture. As an example, the following client-side code fragment writes an informational message to a server's log file:

```
T3Client clnt = new T3Client("t3://acme:7001");
LogServices log = clnt.getT3Services().log();
log.info("Hello from a client");
```

The first line establishes a session with the acme server using the t3 protocol. If RJVMs do not already exist, each JVM constructs an RJVM for the other and an underlying TCP socket is established. The client-side representation of this session - the T3Client object - and the server-side representation communicate through these RJVMs. The server-side supports a variety of services, including database access, remote file access, workspaces, events, and logging. The second line obtains a LogServices object and the third line writes the message.

Clustered enterprise Java™ computer architecture 300 also supports a server-neutral syntax consistent with a peer-to-peer distributed processing architecture. As an example, the following

code fragment obtains a stub for an RMI object from the JNDI-compliant naming service on a server and invokes one of its methods.

```
5      Hashtable env = new Hashtable();  
      env.put(Context.PROVIDER_URL, "t3://acme:7001");  
      env.put(Context.INITIAL_CONTEXT_FACTORY,  
        "weblogic.jndi.WebLogicInitialContextFactory");  
      Context ctx = new InitialContext(env);  
      Example e = (Example) ctx.lookup("acme.eng.example");  
      result = e.example(37);
```

10 In an embodiment, JNDI naming contexts are packaged as RMI objects to implement remote access. Thus, the above code illustrates a kind of RMI bootstrapping. The first four lines obtain an RMI stub for the initial context on the acme server. If RJVMs do not already exist, each side constructs an RJVM for the other and an
15 underlying TCP socket for the t3 protocol is established. The caller-side object - the RMI stub - and the callee-side object - an RMI impl - communicate through the RJVMs. The fifth line looks up another RMI object, an Example, at the name acme.eng.example and the sixth line invokes one of the Example methods. In an embodiment, the
20 Example impl is not on the same processing device as the naming service. In another embodiment, the Example impl is on a client. Invocation of the Example object leads to the creation of the appropriate RJVMs if they do not already exist.

II. Replica-Aware or Smart Stubs/EJB Objects

25 In Fig. 3c, a processing device is able to provide a service to other processing devices in architecture 300 by replicating RMI and/or EJB objects. Thus, architecture 300 is easily scalable and

fault tolerant. An additional service may easily be added to architecture 300 by adding replicated RMI and/or EJB objects to an existing processing device or newly added processing device. Moreover, because the RMI and/or EJB objects can be replicated throughout architecture 300, a single processing device, multiple processing devices, and/or a communication medium may fail and still not render architecture 300 inoperable or significantly degraded.

Fig. 5a illustrates a replica-aware ("RA") or Smart stub 580 in architecture 500. Architecture 500 includes client 504 coupled to communication medium 501. Servers 502 and 503 are coupled to communication medium 501, respectively. Persistent storage device 509 is coupled to server 502 and 503 by communication medium 560 and 561, respectively. In various embodiments, communication medium 501, 560, and 561 may be wired and/or wireless communication mediums as described above. Similarly, in an embodiment, client 504, server 502, and server 503 may be both clients and servers as described above. One of ordinary skill in the art would understand that in alternate embodiments, multiple other servers and clients may be included in architecture 500 as illustrated by ellipses. Also, as stated above, in alternate embodiments, the hardware and software configuration of client 504, server 502 and server 503 is described below and illustrated in Fig. 8.

RA RMI stub 580 is a Smart stub which is able to find out about all of the service providers and switch between them based on a load balancing method 507 and/or failover method 508. In an embodiment, an RA stub 580 includes a replica handler 506 that selects an appropriate load balancing method 507 and/or failover method 507. In an alternate embodiment, a single load balancing method and/or single failover method is implemented. In alternate

embodiments, replica handler 506 may include multiple load balancing methods and/or multiple failover methods and combinations thereof. In an embodiment, a replica handler 506 implements the following interface:

```
5      public interface ReplicaHandler {  
          Object loadBalance(Object currentProvider) throws  
          RefreshAbortedException;  
          Object failOver(Object failedProvider,  
10         RemoteException e) throws  
          RemoteException;  
      }
```

Immediately before invoking a method, RA stub 580 calls load balance method 507, which takes the current server and returns a replacement. For example, client 504 may be using server 502 for
15 retrieving data for database 509a or personal storage device 509. Load balance method 507 may switch to server 503 because server 502 is overloaded with service requests. Handler 506 may choose a server replacement entirely on the caller, perhaps using information about server 502 load, or handler 506 may request server 502 for
20 retrieving a particular type of data. For example, handler 506 may select a particular server for calculating an equation because the server has enhanced calculation capability. In an embodiment, replica handler 506 need not actually switch providers on every invocation because replica handler 506 is trying to minimize the
25 number of connections that are created.

Fig. 6a is a control flow diagram illustrating the load balancing software 507 illustrated in Figs. 5a-b. It should be understood that Fig. 6a is a control flow diagram illustrating the logical sequence of

functions or steps which are completed by software in load balancing method 507. In alternate embodiments, additional functions or steps are completed. Further, in an alternate embodiment, hardware may perform a particular function or all the functions.

5 Load balancing software 507 begins as indicated by circle 600. A determination is then made in logic block 601 as to whether the calling thread established "an affinity" for a particular server. A client has an affinity for the server that coordinates its current transaction and a server has an affinity for itself. If an affinity is
10 established, control is passed to logic block 602, otherwise control is passed to logic block 604. A determination is made in logic block 602 whether the affinity server provides the service requested. If so, control is passed to logic block 603. Otherwise, control is passed to logic block 604. The provider of the service on the affinity server is
15 returned to the client in logic block 603. In logic block 604, a naming service is contacted and an updated list of the current service providers is obtained. A getNextProvider method is called to obtain a service provider in logic block 605. Various embodiments of the getNextProvider method are illustrated in Figs. 6b-g and described in
20 detail below. The service is obtained in logic block 606. Failover method 508 is then called if service is not provided in logic block 606 and load balancing method 507 exits as illustrated by logic block 608. An embodiment of failover method 508 is illustrated in Fig. 7 and described in detail below.

25 Figs. 6b-g illustrate various embodiments of a getNextProvider method used in logic block 605 of Fig. 6a. As illustrated in Fig. 6b, the getNextProvider method selects a service provider in a round robin manner. A getNextProvider method 620 is entered as illustrated by circle 621. A list of current service providers is

obtained in logic block 622. A pointer is incremented in logic block 623. The next service provider is selected based upon the pointer in logic block 624 and the new service provider is returned in logic block 625 and getNextProvider method 620 exits as illustrated by circle 626.

Fig. 6c illustrates an alternate embodiment of a getNextProvider method which obtains a service provider by selecting a service provider randomly. A getNextProvider method 630 is entered as illustrated by circle 631. A list of current service providers is obtained as illustrated by logic block 632. The next service provider is selected randomly as illustrated by logic block 633 and a new service provider is returned in logic block 634. The getNextProvider method 630 then exits, as illustrated by circle 635.

Still another embodiment of a getNextProvider method is illustrated in Fig. 6d which obtains a service provider based upon the load of the service providers. A getNextProvider method 640 is entered as illustrated by circle 641. A list of current service providers is obtained in logic block 642. The load of each service provider is obtained in logic block 643. The service provider with the least load is then selected in logic block 644. The new service provider is then returned in logic block 645 and getNextProvider method 640 exits as illustrated by circle 646.

An alternate embodiment of a getNextProvider method is illustrated in Fig. 6e which obtains a service provider based upon the type of data obtained from the service provider. A getNextProvider method 650 is entered as illustrated by circle 651. A list of current service providers is obtained in logic block 652. The type of data requested from the service providers is determined in logic block 653. The service provider is then selected based on the data type in logic

block 654. The service provider is returned in logic block 655 and getNextProvider method 650 exits as illustrated by circle 656.

Still another embodiment of a getNextProvider method is illustrated in Fig. 6f which selects a service provider based upon the physical location of the service providers. A getNextProvider method 660 is entered as illustrated by circle 661. A list of service providers is obtained as illustrated by logic block 662. The physical distance to each service provider is determined in logic block 663 and the service provider which has the closest physical distance to the requesting client is selected in logic block 664. The new service provider is then returned in logic block 665 and the getNextProvider method 660 exits as illustrated by circle 666.

Still a further embodiment of the getNextProvider method is illustrated in Fig. 6g and selects a service provider based on the amount of time taken for the service provider to respond to previous requests. Control of getNextProvider method 670 is entered as illustrated by circle 671. A list of current service providers is obtained in logic block 672. The time period for each service provider to respond to a particular message is determined in logic block 673. The service provider which responds in the shortest time period is selected in logic block 674. The new service provider is then returned in logic block 675 and control from getNextProvider method 670 exits as illustrated by circle 676.

If invocation of a service method fails in such a way that a retry is warranted, RA 580 stub calls failover method 508, which takes the failed server and an exception indicating what the failure was and returns a new server for the retry. If a new server is unavailable, RA stub 580 throws an exception.

Fig. 7 is a control flow chart illustrating failover software 508 shown in Figs. 5a-b. Failover method 508 is entered as illustrated by circle 700. A failed provider from the list of current providers of services is removed in logic block 701. A getNextProvider method is then called in order to obtain a service provider. The new service provider is then returned in logic block 703 and failover method 508 exits as illustrated by circle 704.

While Figs. 6-7 illustrate embodiments of a replica handler 506, alternate embodiments include the following functions or combinations thereof implemented in a round robin manner.

First, a list of servers or service providers of a service is maintained. Whenever the list needs to be used and the list has not been recently updated, handler 506 contacts a naming service as described below and obtains an up-to-date list of providers.

Second, if handler 506 is about to select a provider from the list and there is an existing RJVM-level connection to the hosting server over which no messages have been received during the last heartbeat period, handler 506 skips that provider. In an embodiment, a server may later recover since death of peer is determined after several such heartbeat periods. Thus, load balancing on the basis of server load is obtained.

Third, when a provider fails, handler 506 removes the provider from the list. This avoids delays caused by repeated attempts to use non-working service providers.

Fourth, if a service is being invoked from a server that hosts a provider of the service, then that provider is used. This facilitates co-location of providers for chained invokes of services.

Fifth, if a service is being invoked within the scope of a transaction and the server acting as transaction coordinator hosts a provider of the service, then that provider is used. This facilitates co-location of providers within a transaction.

5 The failures that can occur during a method invocation may be classified as being either (1) application-related, or (2) infrastructure-related. RA stub 580 will not retry an operation in the event of an application-related failure, since there can be no expectation that matters will improve. In the event of an infrastructure-related failure,
10 RA stub 580 may or may not be able to safely retry the operation. Some initial non-idempotent operation, such as incrementing the value of a field in a database, might have completed. In an embodiment, RA stub 580 will retry after an infrastructure failure only if either (1) the user has declared that the service methods are
15 idempotent, or (2) the system can determine that processing of the request never started. As an example of the latter, RA stub 580 will retry if, as part of load balancing method, stub 580 switches to a service provider whose host has failed. As another example, a RA stub 580 will retry if it gets a negative acknowledgment to a
20 transactional operation.

 A RMI compiler recognizes a special flag that instructs the compiler to generate an RA stub for an object. An additional flag can be used to specify that the service methods are idempotent. In an embodiment, RA stub 580 will use the replica handler described
25 above and illustrated in Fig 5a. An additional flag may be used to specify a different handler. In addition, at the point a service is deployed, i.e., bound into a clustered naming service as described below, the handler may be overridden.

Fig. 5b illustrates another embodiment of the present invention in which an EJB object 551 is used instead of a stub, as shown in Fig. 5a.

III. Replicated JNDI-compliant naming service

5 As illustrated in Fig. 4, access to service providers in architecture 400 is obtained through a JNDI-compliant naming service, which is replicated across architecture 400 so there is no single point of failure. Accordingly, if a processing device which
10 offers a JNDI-compliant naming service fails, another processing device having a replicated naming service is available. To offer an instance of a service, a server advertises a provider of the service at a particular node in a replicated naming tree. In an embodiment, each server adds a RA stub for the provider to a compatible service pool stored at the node in the server's copy of the naming tree. If
15 the type of a new offer is incompatible with the type of offers in an existing pool, the new offer is made pending and a callback is made through a ConflictHandler interface. After either type of offer is retracted, the other will ultimately be installed everywhere. When a client looks up the service, the client obtains a RA stub that contacts
20 the service pool to refresh the client's list of service providers.

 Fig. 4 illustrates a replicated naming service in architecture 400. In an embodiment, servers 302 and 303 offer an example service provider P1 and P2, respectively, and has a replica of the naming service tree 402 and 403, respectively. The node
25 acme.eng.example in naming service tree 402 and 403 has a service pool 402a and 403a, respectively, containing a reference to Example service provider P1 and P2. Client 304 obtains a RA stub 304e by

doing a naming service lookup at the acme.eng.example node. Stub 304e contacts an instance of a service pool to obtain a current list of references to available service providers. Stub 304e may switch between the instances of a service pool as needed for load-balancing and failover.

Stubs for the initial context of the naming service are replica-aware or Smart stubs which initially load balance among naming service providers and switch in the event of a failure. Each instance of the naming service tree contains a complete list of the current naming service providers. The stub obtains a fresh list from the instance it is currently using. To bootstrap this process, the system uses Domain Naming Service ("DNS") to find a (potentially incomplete) initial list of instances and obtains the complete list from one of them. As an example, a stub for the initial context of the naming service can be obtained as follows:

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "t3://acmeCluster:7001");
env.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WebLogicInitialContextFactor");
Context ctx = new InitialContext(env);
```

Some subset of the servers in an architecture have been bound into DNS under the name acmeCluster. Moreover, an application is still able to specify the address of an individual server, but the application will then have a single point of failure when the application first attempts to obtain a stub.

A reliable multicast protocol is desirable. In an embodiment, provider stubs are distributed and replicated naming trees are created by an IP multicast or point-to-point protocol. In an IP multicast

embodiment, there are three kinds of messages: Heartbeats, Announcements, and StateDumps. Heartbeats are used to carry information between servers and, by their absence, to identify failed servers. An Announcement contains a set of offers and retractions of services. The Announcements from each server are sequentially numbered. Each receiver processes an Announcement in order to identify lost Announcements. Each server includes in its Heartbeats the sequence number of the last Announcement it has sent. Negative Acknowledgments ("NAKs") for a lost Announcement are included in subsequent outgoing Heartbeats. To process NAKs, each server keeps a list of the last several Announcements that the server has sent. If a NAK arrives for an Announcement that has been deleted, the server sends a StateDump, which contains a complete list of the server's services and the sequence number of its next Announcement. When a new server joins an existing architecture, the new server NAKs for the first message from each other server, which results in StateDumps being sent. If a server does not receive a Heartbeat from another server after a predetermined period of time, the server retracts all services offered by the server not generating a Heartbeat.

IV. Programming Models

Applications used in the architecture illustrated in Figs. 3-5 use one of three basic programming models: (1) stateless or direct, (2) stateless factory or indirect, or (3) stateful or targeted, depending on the way the application state is to be treated. In the stateless

model, a Smart stub returned by a naming-service lookup directly references service providers.

```
Example e = (Example) ctx.lookup("acme.eng.example");  
result1 = e.example(37);  
5      result2 = e.example(38);
```

In this example, the two calls to example may be handled by different service providers since the Smart stub is able to switch between them in the interests of load balancing. Thus, the Example service object cannot internally store information on behalf of the application. Typically the stateless model is used only if the provider is stateless. As an example, a pure stateless provider might compute some mathematical function of its arguments and return the result. Stateless providers may store information on their own behalf, such as for accounting purposes. More importantly, stateless providers may access an underlying persistent storage device and load application state into memory on an as-needed basis. For example, in order for example to return the running sum of all values passed to it as arguments, example might read the previous sum from a database, add in its current argument, write the new value out, and then return it. This stateless service model promotes scalability.

In the stateless factory programming model, the Smart stub returned by the lookup is a factory that creates the desired service providers, which are not themselves Smart stubs.

```
ExampleFactory gf = (ExampleFactory)  
25      ctx.lookup("acme.eng.example");  
Example e = gf.create();  
result1 = e.example(37);  
result2 = e.example(38);
```

In this example, the two calls to example are guaranteed to be handled by the same service provider. The service provider may therefore safely store information on behalf of the application. The stateless factory model should be used when the caller needs to
5 engage in a "conversation" with the provider. For example, the caller and the provider might engage in a back-and-forth negotiation. Replica-aware stubs are generally the same in the stateless and stateless factory models, the only difference is whether the stubs refer to service providers or service provider factories.

10 A provider factory stub may failover at will in its effort to create a provider, since this operation is idempotent. To further increase the availability of an indirect service, application code must contain an explicit retry loop around the service creation and invocation.

```
15 while (true) {  
    try {  
        Example e = gf.create();  
        result1 = e.example(37);  
        result2 = e.example(38);  
20        break;  
    } catch (Exception e) {  
        if (!retryWarranted(e))  
            throw e;  
    }  
25 }
```

This would, for example, handle the failure of a provider e that was successfully created by the factory. In this case, application code should determine whether non-idempotent operations

completed. To further increase availability, application code might attempt to undo such operations and retry.

In the stateful programming model, a service provider is a long-lived, stateful object identified by some unique system-wide key.

5 Examples of "entities" that might be accessed using this model include remote file systems and rows in a database table. A targeted provider may be accessed many times by many clients, unlike the other two models where each provider is used once by one client. Stubs for targeted providers can be obtained either by direct lookup, 10 where the key is simply the naming-service name, or through a factory, where the key includes arguments to the create operation. In either case, the stub will not do load balancing or failover. Retries, if any, must explicitly obtain the stub again.

There are three kinds of beans in EJB, each of which maps to one of the three programming models. Stateless session beans are 15 created on behalf of a particular caller, but maintain no internal state between calls. Stateless session beans map to the stateless model. Stateful session beans are created on behalf of a particular caller and maintain internal state between calls. Stateful session beans map to 20 the stateless factory model. Entity beans are singular, stateful objects identified by a system-wide key. Entity beans map to the stateful model. All three types of beans are created by a factory called an EJB home. In an embodiment, both EJB homes and the beans they create are referenced using RMI. In an architecture as 25 illustrated in Figs. 3-5, stubs for an EJB home are Smart stubs. Stubs for stateless session beans are Smart stubs, while stubs for stateful session beans and entity beans are not. The replica handler to use for an EJB-based service can be specified in its deployment descriptor.

To create an indirect RMI-based service, which is required if the object is to maintain state on behalf of the caller, the application code must explicitly construct the factory. A targeted RMI-based service can be created by running the RMI compiler without any special flags and then binding the resulting service into the replicated naming tree. A stub for the object will be bound directly into each instance of the naming tree and no service pool will be created. This provides a targeted service where the key is the naming-service name. In an embodiment, this is used to create remote file systems.

V. Hardware and Software Components

Fig. 8 shows hardware and software components of an exemplary server and/or client as illustrated in Figs. 3-5. The system of Fig. 8 includes a general-purpose computer 800 connected by one or more communication mediums, such as connection 829, to a LAN 840 and also to a WAN, here illustrated as the Internet 880. Through LAN 840, computer 800 can communicate with other local computers, such as a file server 841. In an embodiment, file server 801 is server 303 as illustrated in Fig. 3. Through the Internet 880, computer 800 can communicate with other computers, both local and remote, such as World Wide Web server 881. In an embodiment, Web server 881 is server 303 as illustrated in Fig. 3. As will be appreciated, the connection from computer 800 to Internet 880 can be made in various ways, e.g., directly via connection 829, or through local-area network 840, or by modem (not shown).

Computer 800 is a personal or office computer that can be, for example, a workstation, personal computer, or other single-user or

multi-user computer system; an exemplary embodiment uses a Sun SPARC-20 workstation (Sun Microsystems, Inc., Mountain View, CA). For purposes of exposition, computer 800 can be conveniently divided into hardware components 801 and software components 802; however, persons of ordinary skill in the art will appreciate that this division is conceptual and somewhat arbitrary, and that the line between hardware and software is not a hard and fast one. Further, it will be appreciated that the line between a host computer and its attached peripherals is not a hard and fast one, and that in particular, components that are considered peripherals of some computers are considered integral parts of other computers. Thus, for example, user I/O 820 can include a keyboard, a mouse, and a display monitor, each of which can be considered either a peripheral device or part of the computer itself, and can further include a local printer, which is typically considered to be a peripheral. As another example, persistent storage 808 can include a CD-ROM (compact disc read-only memory) unit, which can be either peripheral or built into the computer.

Hardware components 801 include a processor (CPU) 805, memory 806, persistent storage 808, user I/O 820, and network interface 825 which are coupled to bus 810. These components are well understood by those of skill in the art and, accordingly, need be explained only briefly here.

Processor 805 can be, for example, a microprocessor or a collection of microprocessors configured for multiprocessing.

Memory 806 can include read-only memory (ROM), random-access memory (RAM), virtual memory, or other memory technologies, singly or in combination. Persistent storage 808 can include, for example, a magnetic hard disk, a floppy disk, or other

5 persistent read-write data storage technologies, singly or in combination. It can further include mass or archival storage, such as can be provided by CD-ROM or other large-capacity storage technology. (Note that file server 841 provides additional storage capability that processor 805 can use.)

User I/O (input/output) hardware 820 typically includes a visual display monitor such as a CRT or flat-panel display, an alphanumeric keyboard, and a mouse or other pointing device, and optionally can further include a printer, an optical scanner, or other devices for user
10 input and output.

Network I/O hardware 825 provides an interface between computer 800 and the outside world. More specifically, network I/O 825 lets processor 805 communicate via connection 829 with other processors and devices through LAN 840 and through the Internet
15 880.

Software components 802 include an operating system 850 and a set of tasks under control of operating system 310, such as a Java™ application program 860 and, importantly, JVM software 354 and kernel 355. Operating system 310 also allows processor 805 to
20 control various devices such as persistent storage 808, user I/O 820, and network interface 825. Processor 805 executes the software of operating system 310, application 860, JVM 354 and kernel 355 in conjunction with memory 806 and other components of computer system 800. In an embodiment, software 802 includes network
25 software 302a, JVM1, RJVM2 and RJVM3, as illustrated in server 302 of Fig. 3c. In an embodiment, Java™ application program 860 is Java™ application 302c as illustrated in Fig. 3c.

Persons of ordinary skill in the art will appreciate that the system of Fig. 8 is intended to be illustrative, not restrictive, and that a wide variety of computational, communications, and information devices can be used in place of or in addition to what is shown in Fig. 8. For example, connections through the Internet 880 generally involve packet switching by intermediate router computers (not shown), and computer 800 is likely to access any number of Web servers, including but by no means limited to computer 800 and Web server 881, during a typical Web client session.

The foregoing description of the preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications, thereby enabling others skilled in the art to understand the invention for various embodiments and with the various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.

CLAIMS

What is claimed is:

1. A distributed processing system, comprising:
a communication medium;
5 a first processing device, coupled to the communication medium, having a first software program emulating a processing device ("JVM1") including a first kernel software layer, having a data structure ("RJVM2"); and,
a second processing device, coupled to the communication
10 medium, having a first software program emulating a processing device ("JVM2") including a first kernel software layer, having a first data structure ("RJVM1"), wherein a message from the first processing device is transferred to the second processing device through the first kernel layer and first software program in the first
15 processing device to the first software program and the kernel software layer in the second processing device.
2. The distributed processing system of claim 1, wherein the first software program in the first processing device is a Java™ virtual machine ("JVM") and the data structure in the first processing
20 device is a remote Java™ virtual machine ("RJVM") and wherein,
the first software program in the second processing device is a Java™ virtual machine ("JVM") and the data structure in the second processing device is a remote Java™ virtual machine ("RJVM") and wherein the RJVM in the second processing device corresponds to
25 the JVM in the first processing device.

3. The distributed processing system of claim 2, wherein the first kernel software layer in the first processing device includes a socket manager software component.

5 4. The distributed processing system of claim 2, wherein the first kernel software layer in the first processing device includes a thread manager software component.

5. The distributed processing system of claim 2, wherein the RJVM in the first processing device includes a message routing software component.

10 6. The distributed processing system of claim 2, wherein the RJVM in the first processing device includes a message compression software component.

7. The distributed processing system of claim 2, wherein the RJVM in the first processing device includes a peer-gone
15 detection software component.

8. The distributed processing system of claim 2, wherein the first processing device communicates with the second processing device using a protocol selected from the group consisting of TCP, SSL, HTTP tunneling, and IIOp tunneling.

9. The distributed processing system of claim 1, wherein the first processing device includes a memory for storing a Java™ application.

10. The distributed processing system of claim 1, wherein
5 the first processing device is a client and the second processing device is a server for providing a service to the client in response to a client request.

11. The distributed processing system of claim 1, wherein the first processing device is a peer of the second processing device.

10 12. The distributed processing system of claim 10, further comprising:

a second communication medium coupled to the second processing device; and

a third processing device, coupled to the second
15 communication medium, having 1) a software program emulating a processing device ("JVM3"), and 2) a first kernel software layer having a first data structure ("RJVM1"), and a second data structure ("RJVM2").

13. The distributed processing system of claim 12, wherein
20 the second processing device forwards a message from the first processing device to the third processing device.

14. The distributed processing system of claim 13, wherein the first kernel software layer in the first processing device includes an abbreviation table for abbreviating the message and the third processing device includes a duplicate abbreviation table for reading the message.

15. The distributed processing system of claim 13, wherein the third processing device is coupled to the first communication medium, and wherein the message is transferred to the third processing device through the first kernel software layer in the first processing device to the first kernel software layer in the third processing device.

16. The distributed processing system of claim 1, wherein the first processing device includes a stub.

17. The distributed processing system of claim 16, wherein the stub includes a replica-handler.

18. The distributed processing system of claim 17, wherein the replica-handler includes a load balancing software component and a failover software component.

19. The distributed processing system of claim 1, wherein the first processing device includes an Enterprise Java™ Bean object.

20. The distributed processing system of claim 1, wherein the first processing device includes a naming tree having a pool of RA stubs stored at a node of the tree and the second processing device includes a duplicate of the naming tree.

5 21. The distributed processing system of claim 1, wherein the first processing device includes a multicast program for distributing 1) a RA stub to the first and the second processing device, and 2) a naming tree to the first and the second processing device.

10 22. The distributed processing system of claim 1, wherein the first processing device includes an application program coded in a stateless program model.

23. The distributed processing system of claim 22, wherein the application program includes a stateless session bean.

15 24. The distributed processing system of claim 1, wherein the first processing device includes an application program coded in an stateless factory program model.

25. The distributed processing system of claim 24, wherein the application program includes a stateful session bean.

26. The distributed processing system of claim 1, wherein the first processing device includes an application program coded in a stateful program model.

27. The distributed processing system of claim 26, wherein
5 the application program includes an entity session bean.

28. An article of manufacture, including an information storage medium wherein is stored, comprising:

a first set of digital information for transferring a message from a first remote Java™ virtual machine in a first processing device to a
10 second Java™ virtual machine in a second processing device, wherein the first remote Java™ virtual machine includes a message routing software component.

29. The article of manufacture of claim 28, wherein the first remote Java™ virtual machine further includes a message
15 compression software component.

30. The article of manufacture of claim 28, wherein the first remote Java™ virtual machine further includes a peer-gone detection software component.

31. The article of manufacture of claim 28, wherein the first
20 set of digital information further includes a thread manager software component and a socket manager software component.

32. The article of manufacture of claim 28, further comprising:

a second set of digital information for transferring the message to a communication medium.

5 33. The article of manufacture of claim 28, further comprising:

a second set of digital information for transferring the message to a communication medium; and

a third set of digital information, including a remote method
10 invocation.

34. The article of manufacture of claim 28, further comprising:

a second set of digital information for transferring the message to a communication medium; and

15 a third set of digital information including an enterprise Java™ bean.

35. An electronic signal for providing a message from a first computer on a communication medium to a second computer, wherein the electronic signal is embodied in a processor readable memory and includes a first signal section representing a Java™
- 5 virtual machine ("JVM1") and a remote Java™ virtual machine ("RJVM1") for storing in the first computer and a first Java™ virtual machine ("JVM2") and a first remote Java™ virtual machine ("RJVM2") for storing in the second computer, wherein the first
- 10 RJVM1 in the first computer transfers the message to the first RJVM2 in the second computer using a socket.

1/15

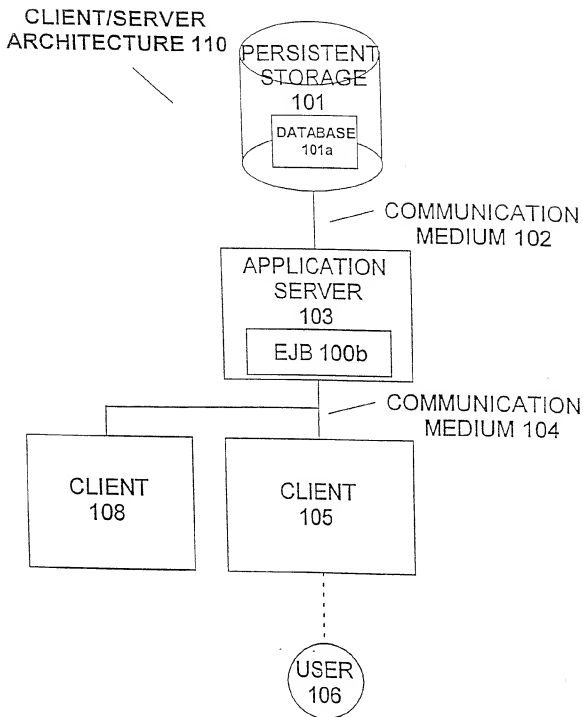


FIG. 1a(Prior Art)

SUBSTITUTE SHEET (RULE 26)

2/15

JAVA ENTERPRISE APIs 100

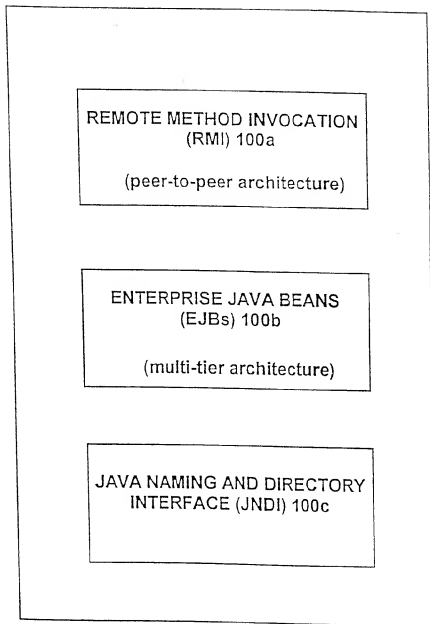


FIG. 1b(Prior Art)

3/15

MULTI-TIER
ARCHITECTURE
160

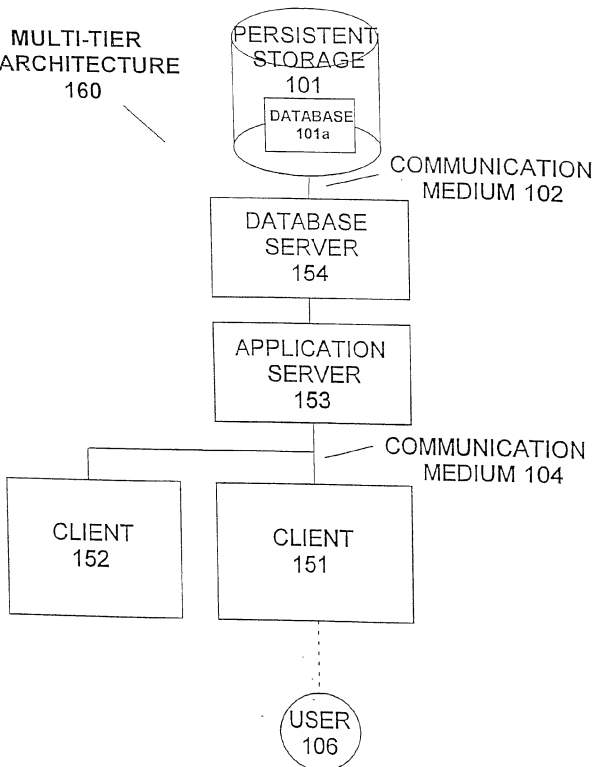


FIG. 1c(Prior Art)

4/15

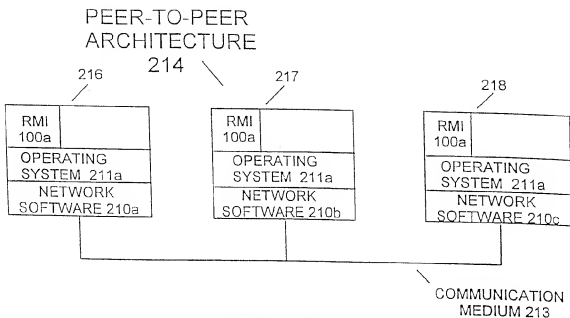


FIG. 2a (Prior Art)

TRANSACTION PROCESSING
(TP) ARCHITECTURE

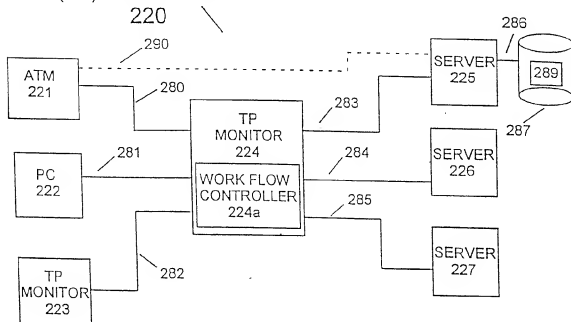


FIG. 2b (Prior Art)

5/15

380

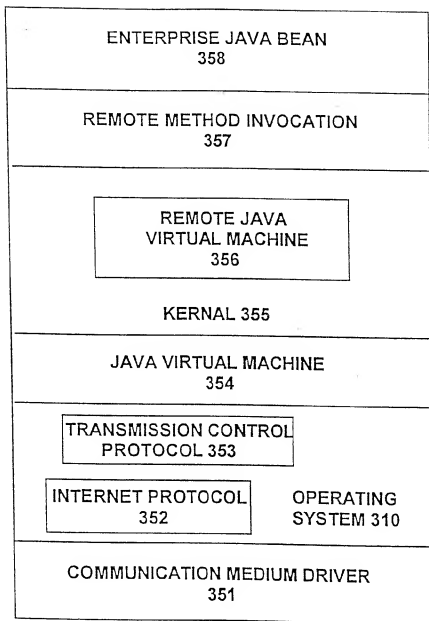


FIG. 3a

6/15

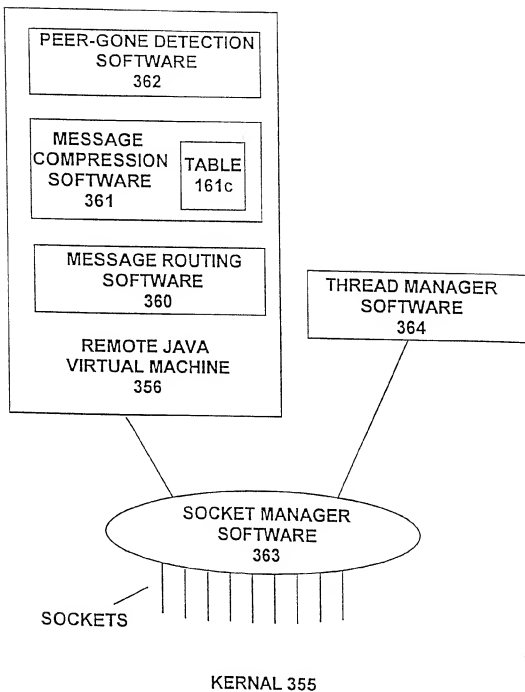


FIG. 3b

SUBSTITUTE SHEET (RULE 26)

7/15

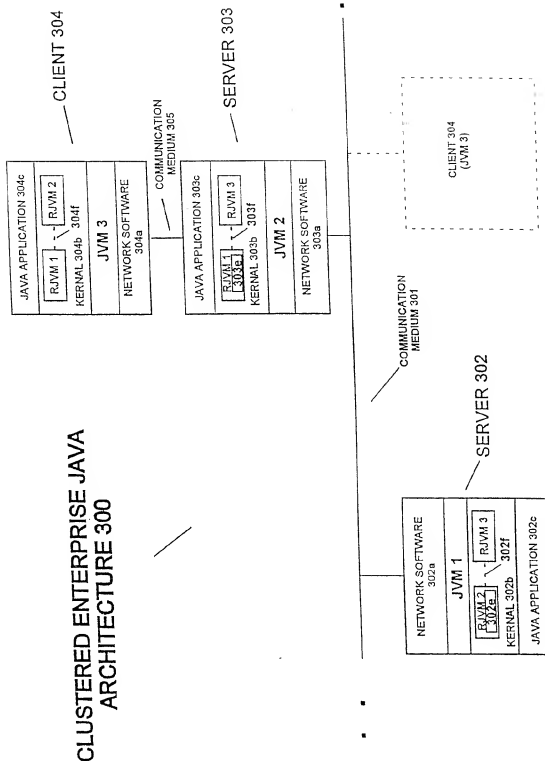


FIG. 3c

8/15

CLUSTERED ENTERPRISE JAVA ARCHITECTURE NAMING SERVICE 400

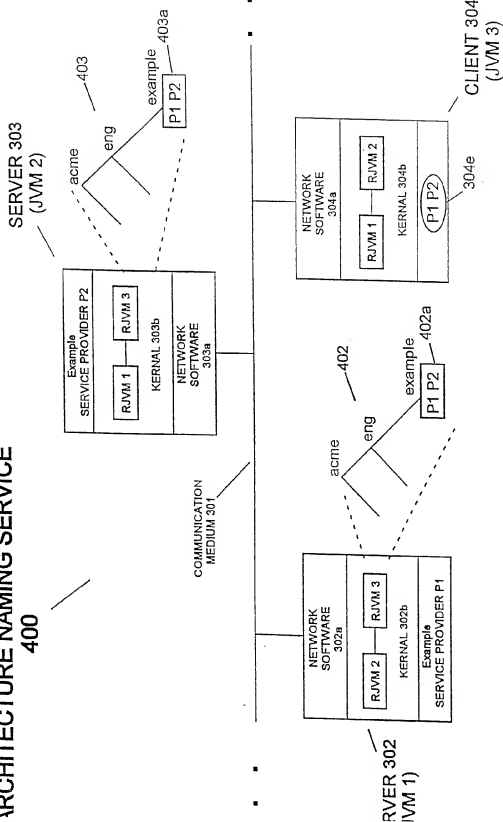


FIG. 4

SUBSTITUTE SHEET (RULE 26)

9/15

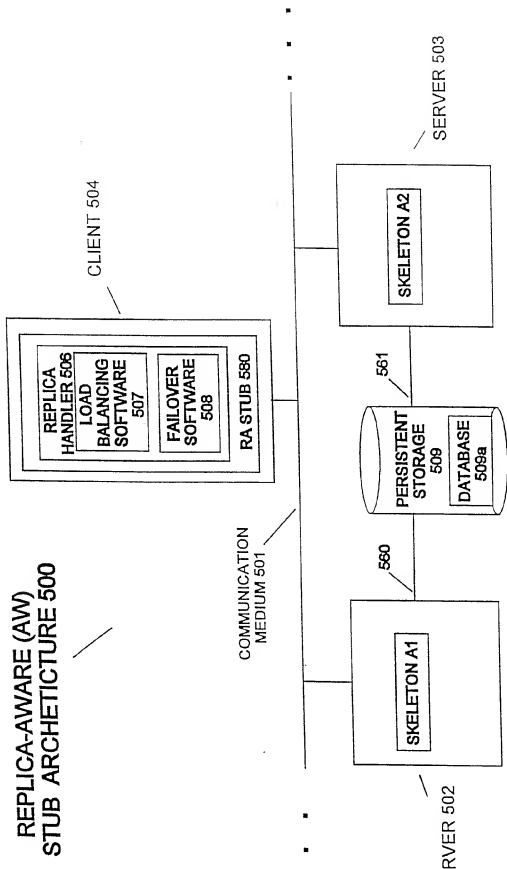


FIG. 5a

10/15

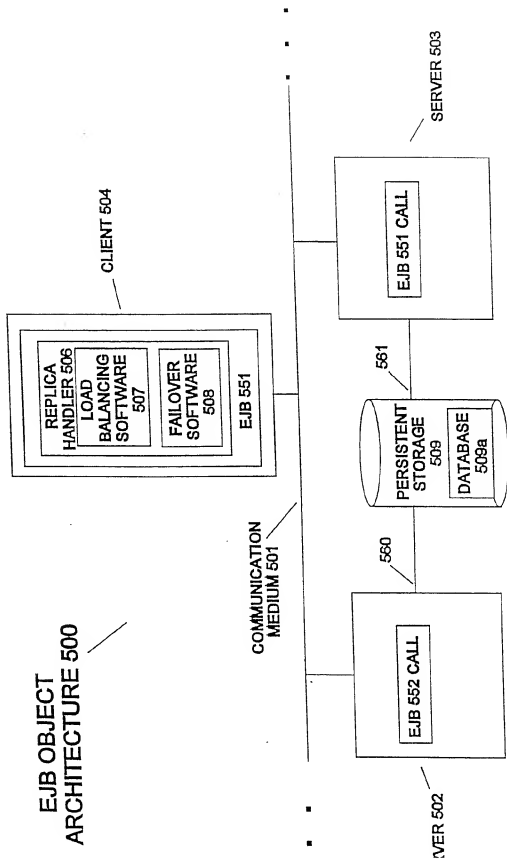


FIG. 5b

11/15

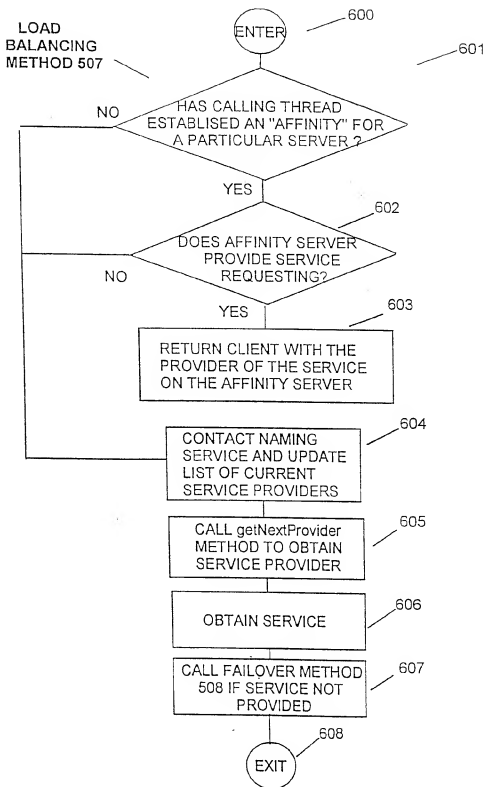


FIG. 6a

12/15

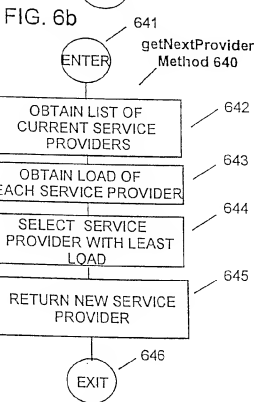
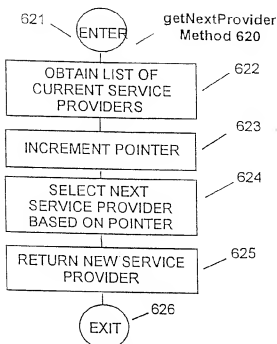


FIG. 6d

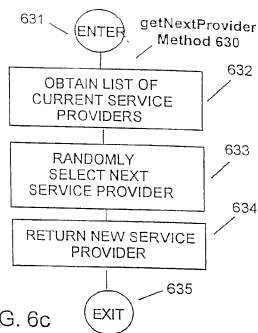


FIG. 6c

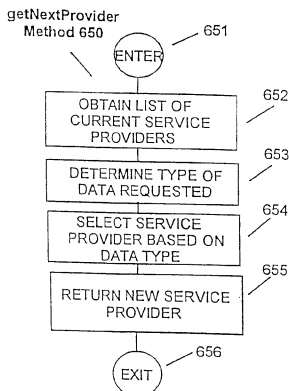
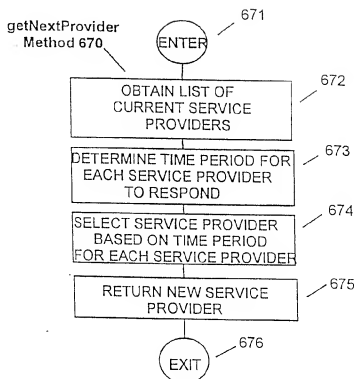
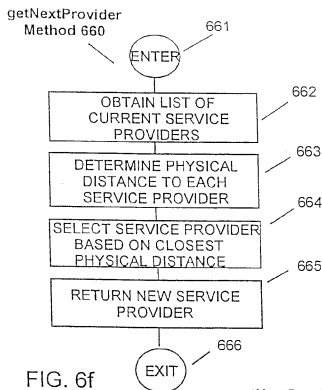


FIG. 6e

13/15



14/15

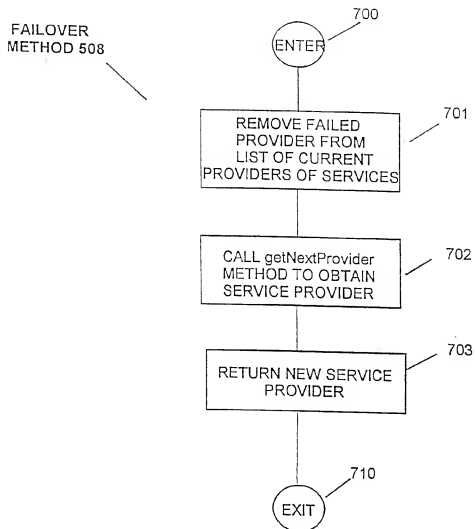


FIG. 7

15/15

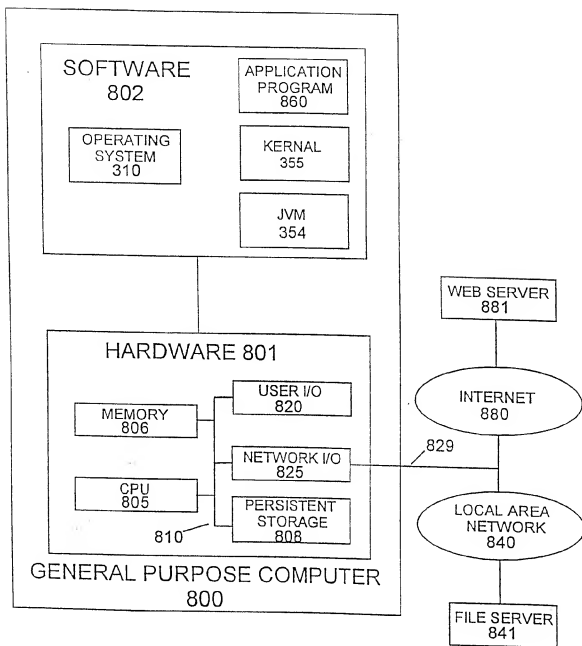


FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/24561

A. CLASSIFICATION OF SUBJECT MATTER IPC(Class) G06F 15/16 US CL. 709/200 According to International Patent Classification (IPC) or to both national classification and IPC				
II. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) U.S. 709/200, 201 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)				
C. DOCUMENTS CONSIDERED TO BE RELEVANT				
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.		
X,E	US 6,003,065 A (YAN et al) 14 DECEMBER 1999, see entire document	1-7,9-15,22,26,28-33,35		
<input type="checkbox"/> Further documents are listed in the continuation of Box C. <input type="checkbox"/> See patent family annex.				
<table border="0"> <tr> <td style="vertical-align: top;"> <p>*A* document defining the general state of the art which is not considered to be of particular relevance.</p> <p>*B* earlier document published on or after the international filing date.</p> <p>*C* document which may throw doubts on priority claims or which is cited to establish the publication date of another document or other special reason (as specified).</p> <p>*D* document referring to an oral disclosure, use, exhibition or other means.</p> <p>*E* document published prior to the international filing date but later than the priority date claimed.</p> </td> <td style="vertical-align: top;"> <p>*F* prior document published after the international filing date or priority date but not in combination with the application not cited to understand the principle or theory underlying the invention.</p> <p>*G* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone.</p> <p>*H* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combinations being obvious to a person skilled in the art.</p> <p>*I* document member of the same patent family.</p> </td> </tr> </table>			<p>*A* document defining the general state of the art which is not considered to be of particular relevance.</p> <p>*B* earlier document published on or after the international filing date.</p> <p>*C* document which may throw doubts on priority claims or which is cited to establish the publication date of another document or other special reason (as specified).</p> <p>*D* document referring to an oral disclosure, use, exhibition or other means.</p> <p>*E* document published prior to the international filing date but later than the priority date claimed.</p>	<p>*F* prior document published after the international filing date or priority date but not in combination with the application not cited to understand the principle or theory underlying the invention.</p> <p>*G* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone.</p> <p>*H* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combinations being obvious to a person skilled in the art.</p> <p>*I* document member of the same patent family.</p>
<p>*A* document defining the general state of the art which is not considered to be of particular relevance.</p> <p>*B* earlier document published on or after the international filing date.</p> <p>*C* document which may throw doubts on priority claims or which is cited to establish the publication date of another document or other special reason (as specified).</p> <p>*D* document referring to an oral disclosure, use, exhibition or other means.</p> <p>*E* document published prior to the international filing date but later than the priority date claimed.</p>	<p>*F* prior document published after the international filing date or priority date but not in combination with the application not cited to understand the principle or theory underlying the invention.</p> <p>*G* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone.</p> <p>*H* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combinations being obvious to a person skilled in the art.</p> <p>*I* document member of the same patent family.</p>			
Date of the actual completion of the international search 25 FEBRUARY 2000		Date of mailing of the international search report 21 MAR 2000		
Name and mailing address of the ISA/US Commissioner of Patents and Trademark Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230 Telex: 250000		Authorized officer FRANK ASTA Telephone No. (703) 305-3847		

INTERNATIONAL SEARCH REPORT

International Application No.

PCT/US 01/47067

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L29/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 1 081 918 A (HEWLETT PACKARD CO) 7 March 2001 (2001-03-07) page 2, line 40 -page 3, line 17 page 3, line 45 -page 4, line 42 figures 1-5 abstract	10,11, 14-17, 19,20, 23-26
Y	---	12,13, 18,21, 22,27
Y	EP 0 991 244 A (NORTEL NETWORKS CORP) 5 April 2000 (2000-04-05) column 1, line 45 -column 3, line 2 --- -/--	12,13, 21,22

☒ Further documents are listed in the continuation of box C.☒ Patent family members are listed in annex.

* Special categories of cited documents

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

T later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

X document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

Y document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

Z document member of the same patent family

Date of the actual completion of the international search

1 August 2002

Date of mailing of the international search report

23/08/2002

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nt.
Fax: (+31-70) 340-3016

Authorized officer

Körbler, G

INTERNATIONAL SEARCH REPORT

International Application No.

PCT/US 01/47067

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication where appropriate, of the relevant passages:	Relevant to claim No.
Y	GRAVESTOCK P.: "Self-Help for Bugs in the Field - A Device-Initiated Upgrade Solution" September 1999 (1999-09), CIRCUIT CELLAR, THE MAGAZINE FOR COMPUTER APPLICATIONS XP002208351 page 2, left-hand column, line 31 -page 3, middle column, line 15 figures 1,2 ---	18,27
A	WO 01 31852 A (BURNETT ALAN MARK ;KEOGH DAVID BRYAN (GB); ROKE MANOR RESEARCH (GB) 3 May 2001 (2001-05-03) page 8, line 20 -page 10, line 18 figure 3 ---	10-27
A	WO 00 67443 A (NILSEN BOERGE ;ERICSSON TELEFON AB L M (SE)) 9 November 2000 (2000-11-09) page 6, line 1 -page 6, line 6 ---	10-27
A	WO 00 28431 A (BEA SYSTEMS INC) 18 May 2000 (2000-05-18) page 16, line 21 -page 17, line 22 -----	10-27

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

XP-002208351

P.D. 00-09-550
P. 1-4 (4)

FEATURE ARTICLE

Peter Gravestock

Self-Help for Bugs in the Field A Device-Initiated Upgrade Solution

Repairing bugs or upgrading firmware after a product is shipped can be quite challenging, especially when the traditional methods (e.g., upgrade disk, FTP download, or visiting engineer) can't be used. With GoAhead's device-initiated architecture, Peter shows us just how easy remote and automatic fixes can be.



In this article, I take you step-by-step through the process of remotely and automatically upgrading the firmware in devices that have been deployed in the field. The solution I outline here can be used by developers of embedded devices and Internet appliances who need to provide field upgrades for their devices.

Developers and manufacturers share the goal of achieving remote programmability while meeting price pressures. Fixing bugs, distributing upgrades efficiently, and creating a vehicle for further sales are all part of that goal. Consider the following situations:

- How do I upgrade my firmware after it has been deployed in the field? I want to perform the upgrade without disturbing the high performance of my product.
- I'd like to add some new product features to my operating system and application software. Some of these systems are installed behind corporate firewalls. Some connect only intermittently to the Internet. Security

is a high priority.

- I want to fix some bugs and we've already shipped. I have deployed thousands of these in devices that are physically hard to access.
- What's the easiest way to get an upgrade out to our customers, when our product doesn't have a disk or CD drive (e.g., cell phones, set-top boxes)?

Has there ever been a hardware or software product that shipped without one bug? Minor, major, catastrophic, or (yes, you guessed it) even Y2K bugs ship with the product.

What kind of solution would enable you to ship your product on time and fix bugs when they arise later? The obvious answer is for the product to upgrade automatically: "Product, heal thyself!"

LIMITED UPGRADE OPTIONS

Options for automatic upgrades so far have been limited and have presented a host of implementation problems. Some manufacturers develop proprietary solutions in-house or use the traditional methods of field upgrading by having customers manually download the upgrade from the web or FTP site, mailing them a disk or CD, sending out an engineer, recalling the product, or doing nothing at all.

There have been some attempts to provide secure upgrades over the Internet in the PC environment. These solutions see the server as the focal point of administrative activity. Clients are added or removed, passwords are changed, and schedules are chosen from the user interface of the server. The server initiates most communication.

History has shown that managing multiple computer systems from the outside is extraordinarily complex. At the level of embedded devices, where the number of discrete systems is in the

millions, these external solutions become completely unwieldy and incapable of providing either the access or capabilities needed to lower support costs or simplify management. An alternative and contrasting solution is to build intelligence into the device enabling the device to manage itself from the inside.

Until now, there has not been an off-the-shelf, device-initiated solution for remotely upgrading operating systems, software, and firmware in embedded systems over the Internet. Recent advances in the embedded market have begun to address this issue.

An ideal upgrade solution would make your embedded devices smarter and your products more reliable, manageable, easier to use, and less expensive to support. An efficient solution would support network devices by automatically ensuring that they have the latest version of software. It would make it easy to securely integrate, deploy, and publish upgrades over the Internet, Intranets, virtual private networks, and dial-up connections.

DEVICE-INITIATED ARCHITECTURE

Making the upgrade process device-initiated is the key to the architecture of the solution outlined in this article. This solution solves some particularly difficult issues, such as firewall penetration, scalability, simplicity, transient connections, efficiency, and so on. Unlike traditional management schemes in which the central console is responsible for tracking all remote devices and initiating contact, in this design the device is autonomous and is responsible for initiating and performing its own upgrades.

A device that can be addressed from an open network is exposed to unauthorized upgrades. With the device-initiated approach, the address of the upgrade server is preprogrammed into the device. The device will only send upgrade requests to the designated server and will only accept upgrades from that



Figure 1—Here's a quick look at the steps of the device-initiated upgrade process.

server.

Typically, servers cannot initiate communication to devices on a network inside a firewall. However, the device-initiated approach presented here ensures that upgrade requests can communicate with the server through firewalls. It uses standard HTTP requests, and most firewalls are configured to allow HTTP requests to the outside world from within the firewall-protected network. With the device-initiated approach, the device sends these requests through the firewall to the server, thus opening the two-way communication channel that is then used for the upgrade transfer.

This article focuses on the device-initiated approach used in an application called GoAhead FieldUpgrader (V.2.0). The application consists of two parts: a development environment for creating device-specific agent software to be embedded in the target device, and a server that publishes upgrades and fulfills device requests.

The agent software is created by the developer using a web-based development environment that runs on Windows NT (and soon on Solaris). The upgrade agent sits between the RTOS and the application to facilitate and speed the integration process. The developer defines one or more components for any device. Each component

contains its server URL, name, version, and polling frequency.

Once developed, this agent on the device polls the upgrade server, which can reside at the location of the network operator, the manufacturer, or any secure site. The agent queries the server for upgrades at predefined intervals. When an upgrade is available, the device initiates the download of the upgrade information and applies it as specified

by the manufacturer. The automatic process is illustrated in five steps in Figure 1.

The agent polls the server at regular, predetermined intervals to query for upgrades. There is an embedded JavaScript extension available so the manufacturer can change the interval. (The embedded JavaScript used in this particular solution is a strict subset of JavaScript developed specifically for use in memory-constrained embedded systems.)

The server receives the request and validates the request with the upgrade policies that the manufacturer has defined. These policies control which remote agents should receive upgrades. The policies are based on criteria that may include host names, domains, schedules, current revision of the requesting device, or desired pace of the rollout. A policy can also check the server activity and request that the agent try later so the server doesn't get bogged down with more requests than the network can handle. Manufacturers can use these policies in conjunction with their own support policy to provide upgrades only to beta customers or only to those customers who have purchased support maintenance agreements. The server then checks for upgrades for that device. If the request is validated, the

server notifies the device that a valid upgrade is available.

The device then initiates a request for the upgrade. The server points to the top directory in the tree containing upgrade files, then recurses and archives all the files to create a single file called the upgrade payload. If the upgrade payload is too large,

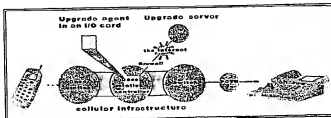


Figure 2—The device-initiated upgrade process can be deployed through a wireless network.

the server divides it into "chunks" so the upgrade can be easily restarted if it is interrupted before completion. If, for example, the server goes down and only three chunks of the upgrade payload have been delivered, the upgrade will continue with the fourth chunk. It will not have to repeat the previous three chunks. The agent then downloads the upgrade payload from the server to the device.

The device authenticates the source of the upgrade using the Diffie-Hellman security keys, reconstructs the payload from the chunks, calculates a 32-bit CRC checksum for each chunk, and performs a higher-level data integrity check using a secure hash algorithm. If the device detects an error in any chunk, it discards that chunk and refetches from the upgrade server. Once all of the chunks have been downloaded and the upgrade has completed successfully, the device notifies the server of the upgrade status.

To give you a better picture of how the pieces fit together in a real environment, Figure 2 depicts where the upgrade server and upgrade agent are deployed in a wireless network.

SECURE UPGRADES OVER THE INTERNET

This solution uses Diffie-Hellman public key exchange to provide encryption keys. The target device and the server each have a public and private key. The device provides its public key to the server, where it is combined with the server's private key to generate a "secret number" that is used to encrypt the upgrade.

Once the target device receives the payload, it decrypts the manifest file. If the decryption is successful, the target device knows that the payload was received from the authorized server. It

then calculates the message digest of the payload and compares it with the value that was sent with the payload. If they match, the target device knows that the payload was not modified en route. If the message digest in the payload doesn't match the number calculated at the target device, the payload is assumed to be suspect and is discarded.

Using this solution, you will no longer need to perform upgrades using FTP to manually retrieve new software, nor send out technicians to visit every remote site to manually perform upgrades.

SAMPLE UPGRADE

Let's walk step-by-step through a sample upgrade of a device. The entire process of creating and running an end-to-end upgrade solution will typically take less than a day.

First, decide what you want to upgrade in the device. For firmware, you can use the upgrade solution in conjunction with your device-specific upgrade utility. If it's the operating system, applications, or data files, the application provides JavaScript with extensions so you can write upgrade scripts that are device-dependent. In Listing 1, you load a DLL, change your upgrade server URL, run the function *doMyUpgrade*, unload your DLL, then reboot the system.

Custom extensions to JavaScript are provided to perform special upgrade tasks. The extensions used in Listing 1 consist of: *loadModule*, *changeServerUrl*, *runFunction*, *unloadModule*, and *reboot*.

To do this task, you need a computer (Pentium class or better) running Windows NT/98 for creation of the agent software using the development environment. You also need a 486 or bet-

ter, running Windows NT to act as the server. In addition, you need:

- RTOS or OS (e.g., VxWorks, Lynx, Windows 95/98/NT)
- TCP/IP stack
- the device in which you want to build upgrade capacity
- GoAhead FieldUpgrader Product Evaluation OR GoAhead UpgradeServer and FieldUpgrader (FieldUpgrader includes the GoAhead DeviceStudio development environment.)

INCORPORATING THE AGENT SOFTWARE

Install and run the development environment on a Windows NT system to create a device-based upgrade agent. Define one or more components for the device. Each component should contain its URL, name, version, and polling frequency. To create the agent:

- use the wizard-based user interface in the development environment to configure the location of your upgrade server (typically an IP address)
- configure the polling frequency. This can be any regular period of time, for example, once a day if you have a small customer base. If you have a large customer base with over 1,000,000 devices, having the devices check for upgrades once a week will spread the network load and allow a single upgrade server to manage the entire device population.
- configure the initial version information for firmware in the device
- select JavaScript if you will be using it
- select the security level you want (you won't need high security if you are working within your own network behind a firewall). Running without security and JavaScript will reduce the memory footprint.

Listing 1—This short listing is an example of some of the custom JavaScript extensions.

```
loadModule("myUpgrade.dll", "myDllOpen");    loads a module(DLL)
changeServerUrl("http://new.server.com");      changes server;
                                                upgrader queries for
                                                upgrades
runFunction("doMyUpgrade");                    runs in-memory function
unloadModule("myUpgrade.dll");                 unloads a module(DLL)
reboot();                                       reboots the system
```

The final step to create the agent is to generate an OS package for your upgrade agent. In Windows NT, for example, this packaging process generates an executable file that is run as an application on the device. In VxWorks, this packaging process generates object files that are linked and compiled within the Tornado development envi-

X. RELATED PROCEEDINGS

None.